コンピュータ・サイエンス研究書シリーズ ⑨

オペレーティング・システム

マサチューセッツ工科大学教授 マドニック/ドノバン著
筑波大学電子・情報工学系教授 池田克夫訳

日本コンピュータ協会

コンピュータ・サイエンス研究書シリーズ

オペレーティング・システム

Stuart E. Madnick / John J. Donovan　著
池田克夫訳

日本コンピュータ協会

Operating Systems

Stuart E. Madnick
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ALFRED P. SLOAN SCHOOL OF MANAGEMENT
and Project MAC

John J. Donovan
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ALFRED P. SLOAN SCHOOL OF MANAGEMENT*
and Project MAC

*Formerly with the Department of Electrical Engineering, M.I.T.

ま え が き

本書の目的

本書は上級ソフトウェア技法について述べてあり，特にオペレーティング・システムに焦点を合せている．本書は，読者が今日のオペレーティング・システムのみならず将来のオペレーティング・システムをも設計したり，利用したり，解析したりすることができるような素材を提供している．内容は組込みの詳細から理論的な結論にわたっている．本書は，例題および演習問題を含んだ教科書として書かれている．

本書の手法

本書の手法は，資源管理の概念に基礎をおいた枠組みを展開しようとするものである．この枠組の中で，例題として，特定のシステムの事例研究を行ない，必要に応じて理論づけを行なっている．

本書は，オペレーティング・システムの設計や研究，および組込みのための総合的な枠組を示している点が特徴である．たいていの教科書では各種の理論を述べたり，事例研究の手法を用いている．前者の場合には，読者はその理論をどこでどのように応用したらよいかがわからず，後者の場合には，根本的な原理と，設計における問題点と，任意に決定してもよい事項との間の区別を行なうことが困難である．

本書の特徴的な点は次の事項である：

1. 本書は１つの科目である．すなわち，単なる記事の寄せ集めではなく，１つのまとまった教科書である．
2. 本書は計算機科学科あるいは情報工学科の学部上級生や大学院生のための教科書である．
3. 各章の終りに演習問題がつけられている．
4. 演習問題の解答と授業時間割りの示された教師用マニュアルが用意されている．
5. 本書は数年間にわたりM. I. T. で行なわれた授業で実際に試験ずみの内容に基づいている．本書はまた Rhode Island 大学や Pittsburgh

大学などの他の大学，および Foxboro 社や米国海軍水産システム・センターなどのいくつかの会社などでも使用されている．

6. 本書は実在のオペレーティング・システムを用いている．とくに，IBM システム/370 から多くの例を引用している．
7. 本書は枠組を作るという手法を用いているので，読みやすく理解しやすい．
8. 実際的な事項を重視している．実際的な動機を示さずに理論的な結論を示した場合，たいていの学生は正しく理解できないことがわかっている．
9. 完全な例題オペレーティング・システムを示している．オペレーティング・システムを学ぶことは自転車に乗ることと似ている．自転車を平衡させる公式を書くことはできても，実際にやってみるまでは自転車に乗ることはできない．
10. 総合的で註釈つきの文献一覧が用意されており，本書で紹介された話題について読者がより詳しく学べるようになっている．
11. 補助教材が用意されている．本書に加えて，次の資料が M. I. T. の計算センターから発行されている．
    a. PL/I で書かれた 360 計算機のシミュレータ
    b. 単純なアセンブラ語の問題に対する採点プログラム
    c. PL/I の問題とこれを実行させる機能
    d. 入出力および割込みの問題
    e. 別のプロセス・スケジューラ（ディスパッチャ）を試験し，実行するための環境シミュレータ
    f. 別の入出力スケジューリング・アルゴリズムを試験し，評価するための入出力環境のシミュレータ
    g. 高速の PL/I コンパイラ——このコンパイラは IBM の PL/I (F) と同様の原始プログラムを入力とし同様の出力を作るが，処理時間が短い．
    h. 学生のジョブを実行するためのサブモニタ

項目11. に示されたプログラムはすべて**ソフトウェア工学のための教育と研究**という題の本に述べられている．各プログラムの説明は次の３つの項目から成っている：

1. 学生のための手引き——パッケージに関して学生に与える情報

2. 教師のための手引き——教師がプログラムを実行するための説明
3. 保守用マニュアル——プログラムの組込みに関する完全な説明

本書の利用法

上に述べたように，本書は学部上級生あるいは大学院生の教科書である．本書を読む前の予備知識としては，アセンブラ，ローダ，コンパイラの原理に関する知識と高水準言語によるプログラミングの経験が必要である．アセンブラ語のプログラミングの経験があることが望ましい．John J. Donovan 著 **システム・プログラム**[1] がこれらの学習に適当である．

教科書として利用する場合には次の4とおりの方法が考えられる：

1. オペレーティング・システムに関する1学期間の集中的な科目．
2. 連続した2学期間の科目．
   a．第一学期：入出力プログラミングと割込み処理の原理を紹介し，演習を行なう．この科目はオペレーティング・システムを将来学ぶための枠組を用意すると同時に，実時間システム，ミニコンピュータ，および特別の応用（たとえば，患者監視用計算機）などのプログラミングの実地訓練にもなる．
   b．第2学期：オペレーティング・システムの概念，設計，組込み，および理論に焦点を合せた上級科目．
3. 本書の一部の章を使用して科目を編成する．
   a．第2章——実時間プログラミング，入出力プログラミング，および割込みの応用に関する科目．
   b．第7章——例題オペレーティング・システムを作成する実験の科目．
   c．オペレーティング・システムに関する概説の科目——第2章の詳細を省略し，第3，第4，第5，および第6章の理論と概念に焦点を合わせる．
4. 専門家養成の科目あるいは独習のため．　　演習問題と解答集を利用する．

   本書は次の点で学生の学習を助けるように意図されている：
   a．オペレーティング・システムの設計
   b．理論の理解

---

1) 池田克夫訳 **システム・プログラム**（日本コンピュータ協会，1974）

c．オペレーティング・システムに用いられた技法を他の分野に応用すること

d．オペレーティング・システムの利用

e．オペレーティング・システムを購入したり評価すること

教科書としては，本書は ACM の COSINE 委員会が最近発表した“オペレーティング・システムの学部科目”という報告で述べている話題を包含している．さらに，ACM の計算機科学科のカリキュラム委員会が**カリキュラム 68**[1]として示した科目のうち次の2つの科目の教材をも含んでいる：

（A 2） 上級計算機構成

（A 8） 大規模情報システム

本教科書は，ACM の経営学科における計算機教育に関するカリキュラム委員会が情報システムに焦点を合せた大学院専門教育のカリキュラム[2]として発表した教材をも含んでいる．これらは次のとおりである：

（C10） 計算機システムとプログラミングの紹介

（C11） 情報構造とファイル（本書の第6章）

（C 2） 計算機システム

（D 2） システム設計

（D 3） システム開発計画（第7章）

第2章では，計算機の構造，機械語によるプログラミング，入出力プログラミング，および割込み処理の基本的事項を示している．オペレーティング・システムを学ぼうとする人は“ビットをいじくる”ことを恐れてはならないと考えている．本章では IBM システム/370 を用いて詳細な例題を示している（計算機の種類に大きく関係しているのは本章だけである）．

第3，第4，第5および第6章ではオペレーティング・システムの解析と設計のための枠組を展開している．本書ではオペレーティング・システムを資源の管理者であるとみなしている．（資源を記憶装置，処理装置，入出力装置，情報の4つに区分している．）これらの章は，資源を管理する技法と問題点とを論じ，最も単純なものから始めて最も複雑なものへと例を用いながら，資源管

1) Communications of the Association for Computing Machinary (CACM), Vol. 11, No. 3, p. 151 (March, 1968).

2) Communications of the Association for Computing Machinary (CACM), Vol. 15, No. 5, p. 363 (May, 1972).

理の枠組を展開している．これらの章で述べた最大の機能（各資源について最も複雑な例のとき）は MULTICS のようなシステムに対応し，最小の機能は IBM OS/360 の基本制御プログラム（PCP）などのシステムに対応する．

第７章では前章までに展開した枠組を用いてオペレーティング・システムを設計している．一般的な方針から出発して，データ・ベースとルーチンの仕様を記述し，最後に実際の組込みのプログラムを示している．

これらすべての資源管理が１つのシステムに組込まれたときは，もはやこれらは独立ではない．たとえば，処理装置管理はシステムでページングやスワッピングの記憶管理を行なっているか否かに関係する．第８章ではこれらの資源管理の相互関係を明らかにし，性能評価のための技法を展開している．

第９章では展開した枠組を用いて各種の実在のオペレーティング・システムを解析し，第10章にはこの分野の文献に註釈をつけて示している．

Stuart E. Madnick
John J. Donovan

謝　辞

本書を書くに当って次の方々の御援助を頂いた.

補助教師の長である Charles A. Ziering に感謝する. 氏は信頼のできる人である. 氏の第4章および第8章に対する功績は特筆すべきものである. 氏は図の確認(および訂正)と解答集の作成を行なってくれた.

補助教師の Paul Gregory, Judy Piggins, Harry Forsdick, Jeff Bunza, Mike Knaur および Suhundra Umarji は演習問題を, David Schwartz は参考文献の整理を, Chad Carpenter および Jeff Buzen は校正を, そして Anne Beer は索引の作成を手伝ってくれた.

第7章は John DeTreville の論文から発展し, Richard Swift によってさらに進展したものである.

初期の頃の科目を受講し建設的な批判を頂いた M.I.T. の学生諸君にも感謝する.

ほとんど1人で本書を編集し, 校正し, 印書をしてくれた Ellen Nangle に感謝する.

Allen Wayne 工芸社の Irene Gunther のすばらしい編集, Nora Braverman の忍耐, および Irving Weinstock の斬新な様式に特に感謝する.

本書を書くに当り経営科学科に気持のよい環境を与えて下さった Sloan School の Michael Schott-Morton 教授と John Little 教授に感謝する.

Stuart E. Madnick
John J. Donovan

目　　次

第 1 章

本 書 の 構 成

本書は次の 2 つのことを目的として書かれている：

1. 上級のソフトウェア技術の基本である非同期動作，割込み処理，オペレーティング・システムとのインターフェース，ハードウェアとソフトウェアの機能分担などを教えること．
2. オペレーティング・システムを設計したり解析したりすることができるような知識を与えること．

最近の計算機のハードウェアはきわめて高性能であり，多くの用途に使用することができる（図 1.1a 参照）．しかし多くの場合ハードウェアだけでは不便で使いづらいものである．この“裸の計算機”を使いやすくするためにオペレーティング・システムが開発され，基礎となるハードウェアを管理し，より使いやすいインターフェースを利用者と利用者プログラムに提供している（図 1.1b 参照）．今日のオペレーティング・システムは計算機の効率のよい運転に不可欠なものとなり，ハードウェアから切り離せないものであると考えられている．

オペレーティング・システムを研究する理由は多数存在するが，特に重要なものは次のとおりである：

1. 特殊目的のために独自のオペレーティング・システムを設計したり，既存のオペレーティング・システムを変更したりしなければならないことがある．
2. たいていの計算機設備において，オペレーティング・システムの選択と組込み機能の取捨選択は主要な決定項目である．
3. オペレーティング・システムは計算機との主たるインターフェースであるから，利用者は目的を達するためにオペレーティング・システムに

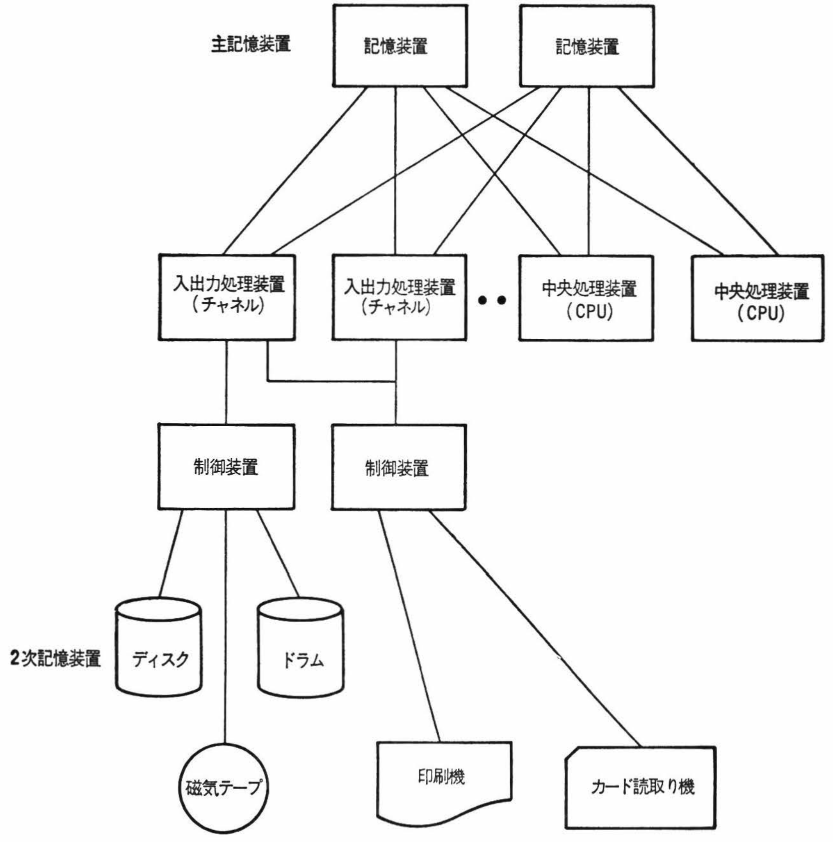


図 1.1a　計算機の基本的なハードウェア（裸の計算機）

仕事を依頼し応答を受ける必要がある．

4.　オペレーティング・システムの中に見出される概念や技法は他の応用分野にも広く適用できるものが多い．

本書は，これらの問題点を述べるのに都合がよいように構成されている．

オペレーティング・システムという言葉は，処理装置，主記憶装置，2次記憶装置，入出力装置，ファイル装置などといった装置を制御するために計算機システム内に用意されたプログラム・モジュールを意味する．

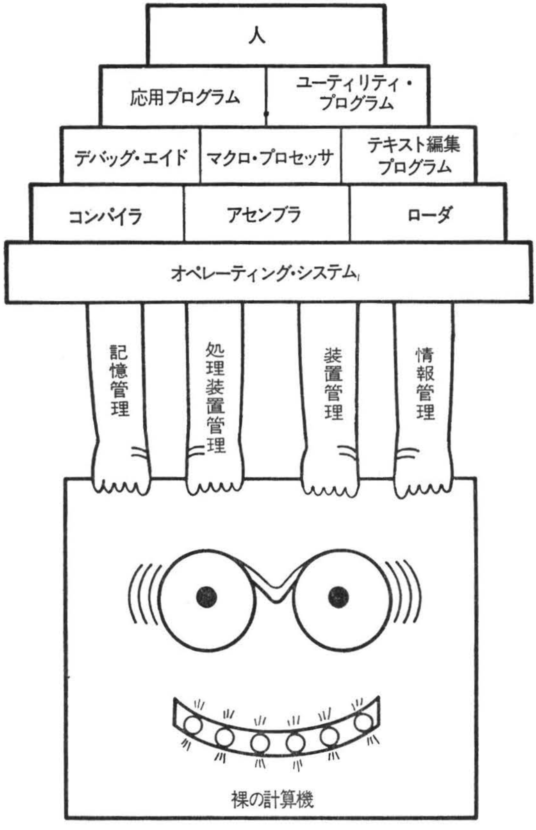


図 1.1b オペレーティング・システムと計算機の基本的なハードウェアの関係

これらのモージュルは矛盾を解決し，効率の最適化を図り，手軽で効果的なシステム利用を可能にする．これらのモジュールは利用者プログラムと計算機ハードウェアとのインターフェースの役割りを果たす．

これらのモジュールを総称して**モニタ**，**制御プログラム**，**スーパバイザ**，あるいは**オペレーティング・システム**などと呼んでいる．応用プログラム，言語処理プログラム，ライブラリ・ルーチン，デバッギング・エイドなどはオペレーティング・システムの定義に含めないことにする．これらは単にオペレーティング・システムの利用者であり，他の書物で述べられている（Donovan, 1972；Knuth, 1968）．

1.1 オペレーティング・システムの重要性

給与計算や銀行業務のような計算量の非常に多い応用分野では，計算機は不可欠なものとなっている．一例をあげると，1950年代の中期にアメリカ銀行は業務に計算機を導入した．もし計算機を利用せず人手で手形の処理をしていたとすれば，カリフォルニア州の成人全部で処理せねばならないところであった(Fano, 1972)．今日計算機は，自動車の設計や医療，宇宙探険などの新しい分野においても使用されている．将来の応用は教育・学習や人工知能の領域にも広がるものと考えられる．

年間200億ドル以上もの金が計算機関係の製品やサービスに支払われている．この金額の70%がソフトウェア開発と保守，データの処理費，およびその他のソフトウェア関係の経費に当てられているとみられている．そして計算機の応用はますます拡大すると思われる．

何が計算機を制御し，利用者と計算機の間の仲介者の役割りを務めるのであろうか．それはほかならぬオペレーティング・システムである．オペレーティング・システムは計算機の動作を制御するプログラムである．たとえば，オペレーティング・システムは計算機利用の計画を立てて仕事を遂行する．高級なオペレーティング・システムは効率を向上させ，その結果計算機を使用する経費を低減させる．たとえば，オペレーティング・システムは1人の利用者の処理結果を出力しながら，同時に，別の利用者のプログラムを実行させる．

本書はオペレーティング・システムの研究，設計および組込みのための枠組を展開しようとするものである．この枠組は小型システム（たとえば第7章で述べる例題システム）および大型システム（たとえば第9章で述べるようなOS/VS）の両方を包含している．

1.2 基本概念と用語

本節は計算機のハードウェア構成，プログラミングおよびオペレーティング・システムの基本概念とこれに関係する用語について述べる．

1.2.1 計算機ハードウェアに関する用語

計算機ハードウェアには多数の異なった方式が存在する．図 1.1aは1つの方式を示す．

命令とデータは**主記憶**，歴史的には**コア・メモリ**と呼ばれるもの（主記憶，

記憶装置およびコアという言葉はしばしば同義語として用いられる）に格納される．記憶装置には各種の処理装置が接続される．**処理装置**とは命令を解釈して指定の動作（たとえば加算）を実行できるハードウェアである．命令とデータは記憶装置の中に符号化されて格納されている．

1つの計算機システムには1台または数台の**中央処理装置**（CPU）がある．CPU は記憶装置に格納されたデータを処理したり，これに算術演算を施す処理装置である． CPU は他の処理装置を制御する命令を実行することもある．CPU が入出力装置の動作を開始させるために入出力開始（SIO）命令を実行できるのはその一例である．

最近のシステムにおいて，ある種の処理装置は特定の処理を効率よく，低廉な経費で実行するために特殊目的専用に構成されている．たとえば，入出力処理装置は入出力装置を制御し，記憶装置と入出力装置の間でのデータの転送を処理する目的で設計されている．この種の処理装置には，特定の1つの入出力装置を制御するものから同時に複数の入出力命令を実行して複数の入出力装置を制御するようなきわめて高度のものまである．

レーザから乳しぼり機に至るおよそ考え得る装置は入出力装置として利用可能であるし，これまでにも利用されている．もっと一般的な装置（カード読取り機，印刷機，ディスク，ドラム，磁気テープなど）の多くは他の装置に対しても適用できるような制御回路を使用している．経済性の理由により，この共通のハードウェアは**制御装置**と呼ばれる独立した装置として入出力装置とは別個に構成されることもある．

計算機システムの中には記憶装置，CPU，入出力処理装置，制御装置，入出力装置などのシステム資源の利用を競合する利用者プログラムが多数存在する．どの利用者がどれだけの期間これらの資源を使用するかを決定するのは何であろうか．入出力装置が動作不能となったときにシステムの運転が停止しないようにしたり，入出力装置に対して入出力処理装置や制御装置を選択し，“情報の経路”を割当てるものは何であろうか．これらの質問のすべてに対する答はオペレーティング・システムである．

1.2.2 プログラミングに関する用語

図 1.1a は計算機システムのハードウェアを示している．たいていの計算機設備においてはハードウェアよりもソフトウェアの方に多くの資金が投入されている．**ソフトウェア**とは仕事を行なうために使用されるプログラムやデータの総称である．**プログラム**は一続きの命令列である（プログラムは時として，

“コード”とも呼ばれる）．プログラムは記憶装置に格納されて処理装置によって解釈（実行）される．プログラムは，使用されないときディスクやドラム，磁気テープなどの2次記憶装置に格納されることが多い．

システムには“かんずめ”になったプログラム（平方根，探索，分類，コンパイラ，アセンブラ，翻訳プログラムなど）が提供されることがある．これらの備えつけのルーチンの管理は誰によってどのように行なわれているだろうか．それはオペレーティング・システムによってである．

図 1.1aに示したような強大なシステムにたった1人で立向ったとしたら意気をくじいてしまうであろう．その人がCPUのプログラミングを習ったばかりのハーバード大学生であり，カードで読込んだ数に49を加算しようとしているものとしよう．どのようにしてカードを読込めばよいだろうか．入出力処理装置のプログラムの仕方も勉強せねばならないのであろうか．いやこれは新米にとっては明らかに荷が重すぎることである．幸いにして，通常，オペレーティング・システムは入出力処理装置のためのルーチンを備えている．一般に，オペレーティング・システムは利用者プログラムの構成と実行の手助けを行なう．

オペレーティング・システムは特定のソフトウェア・パッケージが実行できるような環境を用意する．その1つの例がコンパイラである．コンパイラは高水準言語で書かれた**原始プログラム**（たとえばFORTRANプログラム）を対応する機械語の**目的プログラム**に翻訳する．たいていのコンパイラは，原始プログラムのコンパイルに対しても目的コードの実行に際してもオペレーティング・システムと密接な関係を持っている．コンパイルしている間コンパイラはオペレーティング・システムに原始プログラムの読込み，リストの印刷および出力デックの穿孔を要求する．実行時には，PL/Iなどのコンパイラによって生成された目的コードは複雑な**実行時システム**を必要とする．ALLOCATE文やFREE文はオペレーティング・システムの記憶管理機能の助けを必要とする．PL/IのON CONDITION文はオペレーティング・システムによって用意された割込み処理ルーチンを必要とする．

1.2.3　オペレーティング・システムに関する用語

利用者はジョブをオペレーティング・システムにゆだねる．**利用者**とは計算機システムで何らかの仕事をしたい人のことであり，**ジョブ**とはその仕事を遂行するために必要な一連の処理をいう．ジョブは幾つかのジョブ・ステップに分割される．**ジョブ・ステップ**とは順番に実行しなければならない仕事の単位

をいう．たとえばコンパイル，ロード，および実行は3つのジョブ・ステップである．オペレーティング・システムが利用者のジョブを受付けると，幾つかのプロセスが生み出される．**プロセス**（または**タスク**）とは他の処理と並列的に実行可能の処理をいう． 図 1.2 は利用者，ジョブ，プロセスおよび番地空間の関係を示している．

プロセスの別の定義は，プログラムの集合を実行している処理装置の実行点の軌跡とするものである．1つのプロセスが参照するプログラムやデータの集合は**番地空間**を構成する．図1.2は，例として，1つは入出力プロセス，他はCPU プロセスの2つの番地空間を示している．

注目すべきは，これら2つの番地空間に対するオペレーティング・システムのファイル・システムの部分のコードは，実際には同一の場合もあるということである．このようにコードが共有されるときには“純粋”であるか，あるいは競争条件を排除するためにロックをかけることが必要である．**純粋なコード**（**リエントラント・コード**とも呼ばれる）とは自分自身を修飾しないコードをいう．

オペレーティング・システムはプロセスの番地空間を物理的な記憶域に写像しなければならない．これは特別なハードウェア（たとえばページングを行なうシステム）を用いたり，ソフトウェア（たとえばスワッピング・システム）によって行なわれたりする．

多重プログラミングとは，1つのシステムで，同時に“実行状態”となっている複数のプロセスが存在し得るようにすることをいう．処理が開始されてまだ完了していないか（誤りなどで）終了させられていないとき，プロセスは実行状態である．（**註**：プロセスは実行状態ではあるが実行中でないことがある．すなわち，中間的な結果が得られているが処理装置はこのプロセスの仕事を行なっていない状態である．）

たいていの計算機システムにはオペレーティング・システムのために特別のハードウェアと命令が用意されている．これらの命令は通常の利用者が実行することはできず，**特権命令**と呼ばれている．たいていの計算機は少なくとも2つの実行状態を有している：**プロブレム状態**（**利用者状態**，**スレーブ状態**）および**スーパバイザ状態**（**エグゼクティヴ状態**，**マスタ状態**）である．スーパバイザ状態では処理装置は特権命令を正しく実行することができる．特権命令の1つは計算機の状態を変更するものである．このほかに入出力処理装置を起動したり，記憶装置の保護条件を変更したり，計算機の割込み状態を変更したりする命令がある．

保護機構のハードウェアは記憶装置の部分に対するアクセスを制御する目的

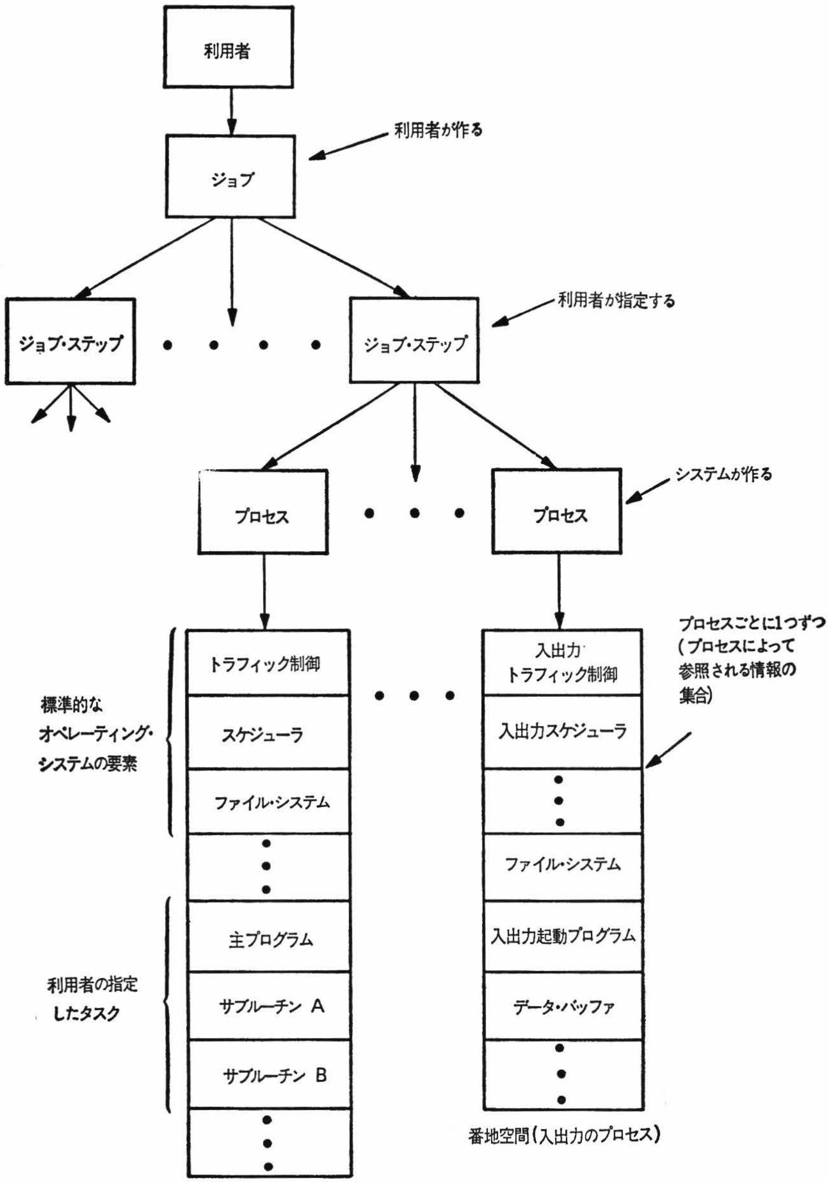


図 1.2　利用者，ジョブ，ジョブ・ステップ，プロセスおよび番地空間

で使用されることが多い．たとえば，オペレーティング・システムは記憶装置のある部分を書込み不可としたいことがある．すなわち，利用者プログラムがオペレーティング・システムのプログラムを変更するのを禁じている．

割込みのハードウェアによって，オペレーティング・システムは同時に遂行される複数の動作を調整することができる．割込み機構によって順次処理的でないプログラムの実行が可能となる．**割込み**とは，処理装置の注意を強制的にその事象に向けさせる機能である．（これはちょうど読者が本書を読んでいるときに親せきや友人に割込まれるのと同じである．）ある種の割込みをマスクするハードウェアが設けられることがある．われわれも時として母親たちを“マスク”することがある．

しかしながら，多くの計算機はこれまで述べた特殊命令やハードウェアのすべてを備えているわけではない．（IBM 1130は一般的なマスク機能を有していない．）これらの機能が欠けているとシステムの利用者や所有者にどんな制限を課すことになるのであろうか．一方，多くの大型システムはオペレーティング・システムのために上述したよりももっと複雑な機能を，特に処理装置において，持っている．これらの特殊機能から利用者は何を得るか．本書はこれら2つの質問に答えようとするものである．

1.3 資源管理者としてのオペレーティング・システム

本書は，オペレーティング・システムは資源の管理者であるという観点に立ってオペレーティング・システムを調べてみようとするものである．

本節ではこれらの資源を論じ，オペレーティング・システムのモジュールがこれらを管理するためになさねばならぬことを定める．第1.4節においてはプロセスから見たこれらの資源の管理の手順を説明するモデルを示す．第1.5節においては階層的な，拡張した計算機という観点からこれらのモジュールの関係と基本機能を説明するためのモデルを示す．

上述の3つの観点は互いに矛盾するものではなく，オペレーティング・システムという1つの対象物の3つの側面を示している．これらの観点は，オペレーティング・システムの理解と研究を助けると同時に，本書の枠組を形成している．使用したモデルは Dijkstra (1968)，Donovan (1972)，Hansen (1970, 1971)，Saltzer (1966)，および Madnick (1969) らを始めとする人々の研究結果の拡張である．

システムはシステムの資源——記憶装置，処理装置，入出力装置および情報

（プログラムおよびデータ）——を管理するために設計されたプログラム（アルゴリズム）の集合体であるというのが本書で採用している基本的な観点である．これらの資源はどれも貴重なものであり，これらが効率よく利用され，しかも利用者間に起こる競合によって矛盾が生じないようにするのがオペレーティング・システムの役目である．

オペレーティング・システムは各資源の状態をつねに把握し，どのプロセスが（どれだけの量と期間）資源を利用できるかを決定し，それを割当て，終了したらそれらを取戻さなければならない．

記憶装置，処理装置，入出力装置（ディスク，磁気テープ，テレタイプ）および情報という資源はその特性が本質的に異なっているので，これらの間の区別を明確にしておかなければならない．上述のものはそれぞれ1つの装置であり，多くの類似点を有してはいるが，このような抽象化は資源管理の理論や技法を展開していくのには役に立たないようである．

オペレーティング・システムを資源の管理者であると考えると，各管理者は次のことを行なわねばならない：

1. 資源の状態をつねに把握すること．
2. 資源の分配の方針を決定すること．
3. 資源を割当てること．
4. 資源を取戻すこと．

オペレーティング・システムに含まれるプログラムを4つに分類する．次に示したのが各類の主要機能とこの機能を果たすルーチンにつけられている典型的な名前である．

1.3.1 記憶管理機能

1. 記憶装置の状態を把握する．どの部分が誰によって使用されているか．どの部分が空いているか．
2. 多重プログラミングが行なわれているときには，どのプロセスがいつ，どれだけ記憶装置を使うかを決定する．
3. プロセスが要求したとき記憶装置を割付ける．2.の方針によってこれを行なう．
4. プロセスに記憶装置が不要になったときやプロセスが終了したとき記憶装置を解放する．

1.3.2 処理装置の管理機能

1. 処理装置とプロセスの状能を把握する．これを行なうプログラムは**トラフィック制御**と呼ばれる（Saltzer, 1969).
2. どのプロセスに処理装置を使用する機会が与えられるかを決定する．**ジョブ・スケジューラ**はシステムに**提出**されたジョブの中でどれをシステムに入れたらよいか，すなわち，どれに資源を割当てたらよいかを決定する．多重プログラミングのときには，どのプロセスにいつどれだけの期間処理装置を割当てるかを決定する．これは**処理装置スケジューラ**と呼ばれる．
3. 必要なハードウェアのレジスタを設定したうえで処理装置をプロセスに割当てる．これは**ディスパッチャ**と呼ばれる．
4. プロセスが処理装置を放棄したり，終了したり，割当量を超過したときに処理装置を解放する．

どのジョブをシステムに入れるかを決定するには数多くの要因を考慮する必要がある．たとえば，システムが所有しているよりも多くの記憶域やテープ駆動装置を要求するようなジョブはシステムに入れることが認められない．このようなジョブには資源を割当ててはならないのである．ジョブ・スケジューラが記憶管理や装置管理の一部分でない理由は何であろうか．歴史的には，1つのジョブをシステムに入れるかどうかを決定する主要因はそれに割当てる処理装置使用時間であったからである．さらに，ジョブ・スケジューリングと処理装置スケジューリングのための記録の管理が両者共に類似している．したがって，これらのモジュールを処理装置管理に含めている．

1.3.3 装置管理機能

1. 入出力装置，チャネル，制御装置の状態を把握する．この機能は**入出力トラフィック制御**と呼ばれる．
2. 装置を効率よく割当てるための方針を決定する．共有装置の場合にはどのプロセスにどれだけの量を与えるかを決定する．この機能を**入出力スケジューリング**と呼ぶ．
3. 装置を割当て入出力動作を開始させる．
4. 装置を解放する．たいていの場合入出力は自動的に終結する．

装置をいかに割当てるかはその装置がどの程度融通がきくかに依存する．あ

る種の装置（たとえば，カード読取り機）は共有することができないので1つのプロセスに**専用**させる必要がある．**共用**できるものもある（たとえば，ディスク）．このような装置はより融通性に富むが，その割当ては複雑である．**仮想的な装置**とすることのできるものもある．たとえば，カードに穿孔する仕事はディスク（仮想的なカード穿孔機）に“書出す”ことに変換することができる．後で，情報をカード穿孔機に複写してやればよい．仮想的なカード読取機，カード穿孔機および印刷機はスプーリング（SPOOL）・ルーチンを用いてその仕事が遂行される．仮想的な装置を導入すると

1. 専用装置を共用することが可能となり，これらの装置のスケジューリングの自由度が増し，
2. ジョブ・スケジューリングの自由度が増し，そして
3. 装置の速度とその装置に対する要求の速度との整合がよくとれるようになる．

1.3.4 情報管理機能

1. 情報の管理を行なう．たとえば格納位置，使用状況，状態など，を把握する．この機能は**ファイル・システム**と総称される．
2. どのプロセスが情報を使用するかを決定し，情報の保護を行ない，アクセス・ルーチンを用意する．
3. 情報を割当てる．ファイルを**オープン**することなど．
4. 情報の割当てを解消する．ファイルを**クローズ**することなど．

1.3.5 資源管理のまとめ

オペレーティング・システムの資源管理機能はすべて上述の4種類に分類できる．現用のオペレーティング・システムの解析を行なったり，新しいオペレーティング・システムを設計したり，これに関する理論を研究したりするためにこの考え方を用いることができる．

第3，第4，第5および第6章においては，オペレーティング・システムを端から順に全体にわたって，これらの4つの資源を詳しく調べてみることにする．第2章では特定の局所的な問題を取扱うことにする．

本書ではオペレーティング・システムを4つの資源――記憶装置，処理装置，入出力装置および情報――の管理という観点から調べることにするので，これらの資源の相互の関係をはっきりさせておくことは重要である．簡単のため

に，最初のうちは，これらの資源の管理は互に独立であるとみなされるが，理解しておかなければならない微妙な相互関係が存在する．第8章ではこれらの関係の幾つかを論じる．

資源とオペレーティング・システムによる資源の管理は家屋を建てるための材木とれんが，釘などの関係と類似している．これらの組合せ方には多数の変形がある．あるものは不細工で，あるものはエレガントである．第1.5節はその1つの組立て方を示している．このような観点に立つオペレーティング・システムの例は第7章で論じられる．例を示すだけではなく，第7章はオペレーティング・システムの組立て方を，他の章では示し得なかった詳細なレベルで論じている．第2章で計算機の構造の例を示している以外は，本書は計算機の構成とは独立である．特定の計算機の例は概念を説明する目的でのみ使用されている．

1.4 プロセスという面から見たオペレーティング・システム（資源管理が働いているところ）

前節では，オペレーティング・システムは資源を管理するためのプログラムの集合体で構成されていると述べた．これらのプログラムの間にはどのような関係があるのだろうか．プロセスの生涯の間のどのようなときにこれらのプログラムが働くのであろうか．この関係と資源モジュールの間の実行順序を説明するために，本章ではプロセスの生涯を調べてみよう．

1つの例を使用してこれを調べてみる． 図 1.3 は多重プログラミング・システムの中に共存する3つのプロセス（利用者のジョブ）を示している．

プロセスの生涯のサイクルは 図 1.4 に示した3つの状態の間の遷移により表現される：

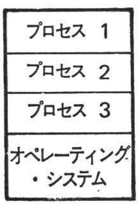


図 1.3 多重プログラミング・システムにおける複数のプロセス

実行中：プロセスに処理装置が割当てられており，プログラムは現在実行中である．

待合せ：プロセスはある事象（たとえば入出力動作の完了など）を待合せている．

実行可：プロセスは“実行できる”状態であるが，処理装置の数よりも多くのプロセスが存在するときには処理装置割当ての番が回ってくるのを待たなければならない．

図 1.4 には状態遷移の原因となる例を示している．たとえば“実行中”のプロセスが入出力完了を待合せる必要があれば“待合せ”状態とならなければならない．

図 1.4 に示したプロセスの状態は簡略化してある．図ではプロセスはすでにシステムに存在しているとし，永久に実行し続けると仮定している．もっと真実味のある完全なプロセスの生涯は 図 1.5 に示されている．

図では 3 つの状態を追加した：

提出：利用者はシステムにジョブを提出する．システムは利用者の要求に応答せねばならない．

保持：利用者のジョブは計算機が読込める内部形式に変換されている．しかしプロセスにはまだ資源が割当てられていない（ジョブはディスク

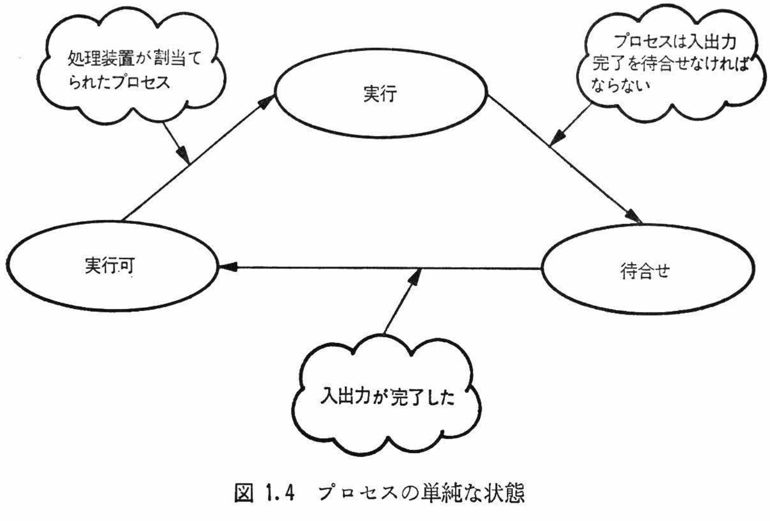


図 1.4　プロセスの単純な状態

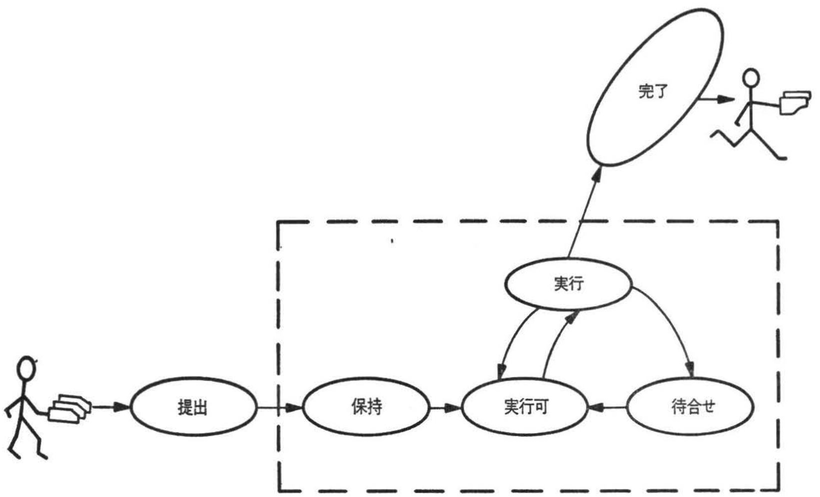


図 1.5 プロセス状態のモデル

上にスプールされている). プロセスを実行可とするには資源を割当てる必要がある.

完了：プロセスは処理を完了し，プロセスに割当てられている資源を返却してもよい.

一般に，オペレーティング・システムは，1つのプロセスがこれらをすべての状態を経て完了するように制御する. 状態と遷移は図 1.6 に示されている. 図で円は状態を示し，雲形はプロセスがある状態から他の状態に切換わるときに呼ばれるオペレーティング・システムのルーチンを示している. システムの中の各プロセスに対しては全く同一の図が描かれる.

図 1.6 でプロセス管理はプロセス状態変化の順序づけに重要な役割りを果たしていることが示されている. ここでは例題を用いて図 1.6 をたどってみよう. 1人の利用者は，カード読取り機にカード・デックをかけてシステムにジョブを提出する (**提出状態**). このジョブはジョブ制御カードに引き続く幾つかのプログラム・デックから構成されている. ジョブ制御カードは，このジョブが必要とする資源に関する情報をオペレーティング・システムに与える. スプーリング・ルーチンはジョブを読込み，ディスクに格納する (**保持状態**). スプー

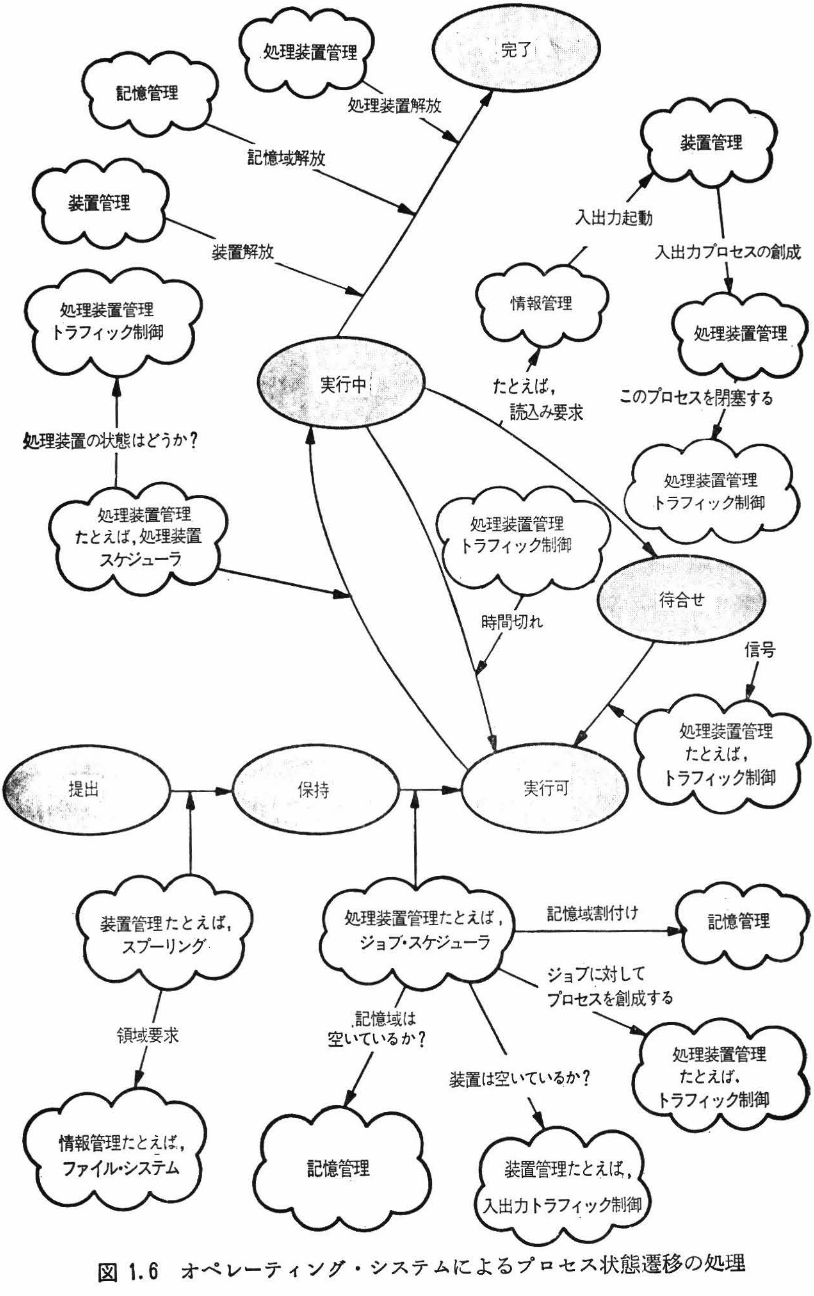


図 1.6 オペレーティング・システムによるプロセス状態遷移の処理

リング・ルーチンは，利用者ジョブを機械語形式に変換したものを格納する領域を求めるために，情報管理を呼出さなければならない．情報管理はシステムの中のすべての情報と利用可能な領域に関する状態を把握している．

後刻，ジョブ・スケジューリング・ルーチンは，システムに入れるために，ディスクにスプールされたファイルを走査してジョブを選び出す．ジョブを選ぶときに，スケジューラは記憶管理を呼出して必要となる記憶領域が得られるかどうかを決定する．次に装置管理を呼出して，ジョブによって要求されている入出力装置が使用できるかどうかを決定する．ジョブ・スケジューラは入出力装置の要求をジョブ・カードから決定する．ジョブ・スケジューラがこのジョブに資源を割当てるべきであると決定するとトラフィック制御が呼出されてプロセス情報を生成し，記憶管理が呼出されて必要量の主記憶を割当てる．ここでジョブは記憶装置にロードされプロセスは実行可能の状態（**実行可状態**）となる．

処理装置が空くとプロセス・スケジューラは実行可プロセスの一覧表を走査し，1つのプロセスを選択し，それにプロセッサを割当てる（**実行状態**）．

実行中のプロセスが情報の参照を要求すると（たとえばファイルの読込み），情報管理は装置管理を呼出してファイルの読込みを開始させる．装置管理は入出力動作を開始してからプロセス管理を呼出し，“ファイルを要求しているプロセスは入出力動作の完了を待合せている”旨通知する（**待合せ状態**）．

入出力が完了すると入出力ハードウェアはプロセス管理に含まれるトラフィック制御に信号を送り，プロセスは実行可の状態に戻される．

プロセスが再び実行状態となり処理を完了したときには完了状態となり，割当てられていた資源はすべて解放される．

図1.6のプロセスの状態は1つのプロセスの生涯の例である．もっと複雑なオペレーティング・システムではもっと多くの状態を定義した方がよいことがある（第4章参照）．一方，オペレーティング・システムがプロセスの概念を用いずに設計されているときには，観測者がプロセスの状態を区別することは困難であろう．

1.5 階層的な拡張した計算機として見たオペレーティング・システム

前節においては，オペレーティング・システムによって管理されるシステムの資源を識別する必要があることを述べた．そしてプロセスという面から見て

いつ資源管理が仕事を行なうかを説明した．さて，次に“資源管理は**どのようにして**呼出され，**どこに**納められているか”という問に答えなければならない．スケジューラは記憶管理のサービスを求めてそれを呼出すのであろうか．記憶管理はスケジューラを呼出すであろうか．プロセスの概念は利用者のみに対するものであろうか．それともオペレーティング・システムに対しても用いられているのであろうか．本節はオペレーティング・システムの階層的な見方を示し，モジュールの関係を示す．これらのモジュールを構成するための方法は絶対的に正しいとか誤っているとかはいえない．しかし，階層的にシステムを考えると，簡潔で，解析したり組込んだりしやすい構造を与える．

1.5.1 拡張した計算機の概念

第1.2.2節では計算機の基本的なハードウェアの特徴を紹介した．これはソフトウェアという衣服を着ていない**裸の計算機**である．たいていのプログラマにとっては裸の計算機は望ましい姿ではない．次のようなプログラムを書きたくなったとしよう：

1	MOVE C,B	C=B と設定
2	FIND-SPACE 80,X	80バイトの空き領域を求めてX に番地を設定
3	READ-CARD X	カードを領域X に読込む
4	COMPARE X(2),'/*'	領域 Xの最初の2バイトは “/*”か？
5	TRANSFER-MATCH END	もしそうならENDへ行く

文1，4および5はシステム/370 などの計算機の命令にきわめてよく対応している．これに対して，文2および3は，記憶装置や入出力装置といった基本的なシステム資源の利用にかかわることであるので，計算機で正しくしかも効率よく遂行しようとすると 10，100，いや 1,000 もの命令が必要である．

これらの種類の資源管理機能を構成する命令はオペレーティング・システムによって用意される．利用者のプログラムは，特別の**スーパバイザ呼出し**命令を用いてこれらのサービスを要求することができる．スーパバイザ呼出し命令はサブルーチン呼出しと非常によく似た動作を行なうが，利用者のサブルーチンではなくてオペレーティング・システムに制御を移す．このようにして，利用者にとって文2および3は，裸の計算機の命令には直ちに対応するわけではないが，拡張した計算機においては正しい命令である．オペレーティング・シ

ステムは基本的なハードウェア命令のほかに多くの“命令”を用意する．これらの命令は総称して**拡張した計算機**の命令と呼ばれる．この関係は 図 1.7 で示されている．オペレーティング・システムの核は裸の計算機で実行され，利用者プログラムは拡張して計算機で実行される．

1.5.2 階層的な計算機の概念

オペレーティング・システムはどのようにして構成されているのであろうか．初期のオペレーティング・システムの大半は単に1つの大きなプログラムから成っていた．システムがだんだん大きくなり，より多くの要素を含むようになるとこのような“腕力法”では対処できなくなってきた．拡張した計算機という考え方はオペレーティング・システムに対して2とおりに適用できることが明らかになってきた（図 1.8 参照）：

1. 多くのシステム・モジュールが必要とする基本機能は“内部的な拡張計算機”に分割することができる．

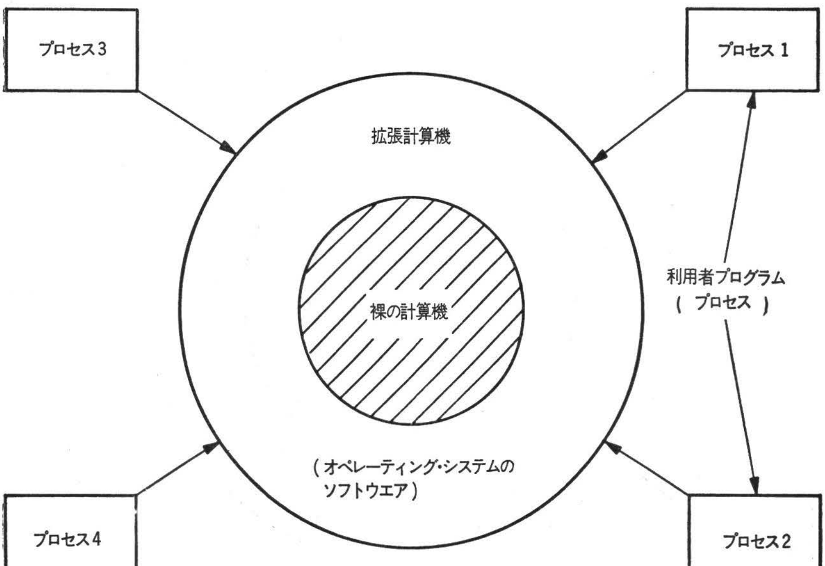


図 1.7 拡張計算機

2. 特定のモジュールは，本質的にはプロセスと同様に，分離することができ，拡張した計算機で実行することができる．

階層的な考え方を採用するには基本的な質問に答えておかねばならない．オペレーティング・システムの各モジュールは**どこに**属するのか．内部の拡張計算機にか，外部の拡張計算機にか，またはプロセスとしてか．さらに，内部／外部の拡張計算機の概念は**拡張計算機の階層**という概念に一般化することができ，また，オペレーティング・システムのプロセスは相互に関係し合い，**プロセスの層**という概念に一般化できる．拡張計算機の中のシステム・モジュールは，プロセスの層として動作するものに対比して，総称して**オペレーティング**

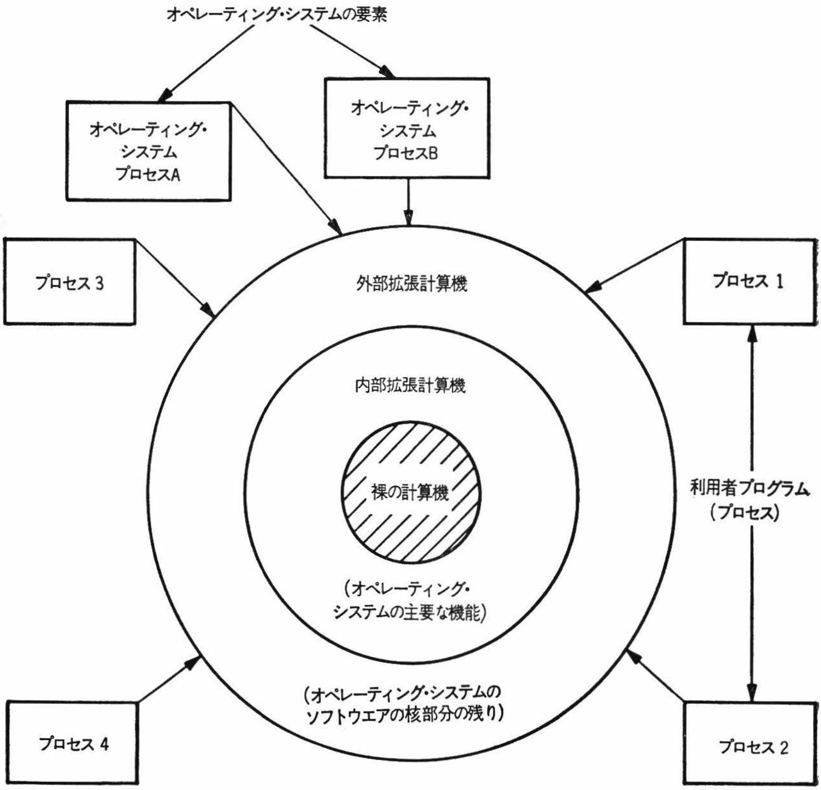


図 1.8　単純な階層的計算機

・システムの核と呼ばれる．現在のところ，層は何層あればよいかとか，1つのモジュールがどの層に入るべきかとか，核にはどのモジュールを入れればよいかは，はっきりした規準がない．

階層化の概念によって最初に設計を行なったのは Dijkstra (1968) の T.H.E. オペレーティング・システムと Madnick (1969) のモジュラ・ファイル・システムである．同様な方法は複雑な応用プログラムの設計において**モジュラ・プログラミング**とか**構造化されたプログラミング**という呼び方で採用されようとしている．本節と第7章の例題のオペレーティング・システムの設計とにおいては，明白で基本的な機能を選んで核部分とし，可能な限りオペレーティング・システムの仕事を別個のシステム・プロセスに分割した．図1.9は本書で論ぜられる階層的構造を示している．すべてのプロセス（図1.9では箱で示されている）は核を使用し，システムの資源を共用する．あるプロセスは，他のプロセスの“親”であったり“制御者”であったりする．この関係を，分割したプロセスの層を示す波状の線によって示している．

厳密に階層化された組込み法では，1つのレベルからは下のレベルのサービスを要求することが許されるが，上のレベルのサービスは要求できない．たとえば，記憶管理モジュールのサービスを要求する必要のあるプロセス間メッセージ管理を設計する場合，この記憶管理はメッセージ管理よりもレベルが下でなければならない．記憶管理からメッセージ管理のサービスを要求することは許されない．図1.9においては，これらのモジュールをそれぞれレベル2およびレベル3に入れて示している．

最低のレベル1におくべき機能は何であろうか．それはすべての資源管理で使用されるものであり，資源の割当ての状態を把握するものである．資源割当ては同期させることが必要である．これらの基本的なオペレータは時として**Pオペレータ**（資源は捕捉されたり要求されたりしていることを示す）および，**Vオペレータ**（資源は解放されていることを示す）と呼ばれる．これらの同期は**セマフォア**（腕木信号）と呼ばれるソフトウェアのからくりによって動作する．

このセマフォアはスイッチのように単純なもの（2値信号）であったり，整数値（計数信号）を持ったりする．システムにおける各資源に対応づけて1つのセマフォアが用意されている．資源が要求されるとPオペレータが使用されて対応したセマフォアを調べる．もしオフであれば，Pはこのセマフォアをオンとして復帰する．フラグがすでにオンであれば，Pは，“要求を行なっているプロセスがそのセマフォアに対応した資源を待合せて待ち状態になった”こと

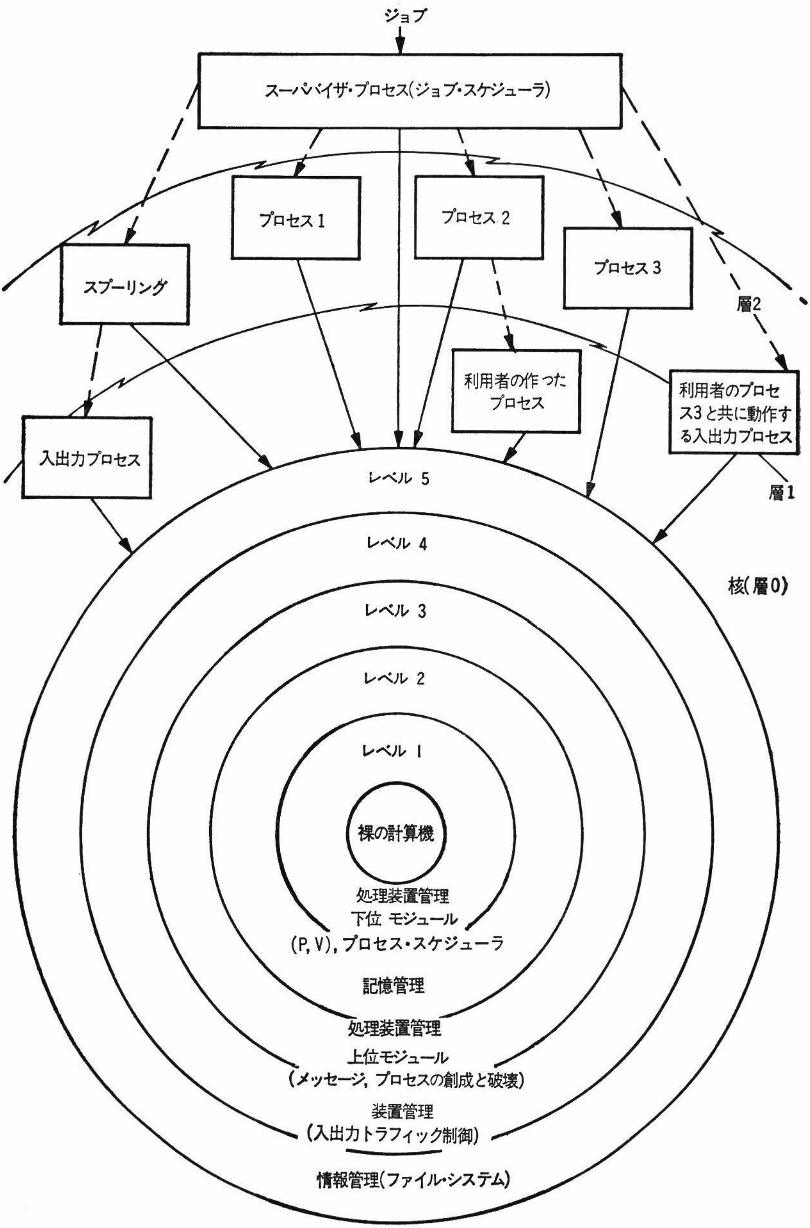


図 1.9 オペレーティング・システムの階層構造

を通告する．後刻，他のプロセスによってVオペレータが実行されると資源が解放され，セマフォアがオフとされ，その資源を待合せていたプロセスが起動される．

第4章においてプロセスの同期はより形式的に説明される．第7章では組込みの例が示される．オペレーティング・システムのたいていのモジュールはPやVオペレータを使用する必要があるので，これらの機能は核の中心に（レベル1）おかれることに注目しなければならない．

核の各レベルにおける基本機能の例は：

レベル1：処理装置管理　低レベル
P同期基本機能
V同期基本機能
プロセス・スケジューリング——多重プログラミング機構

レベル2：記憶管理
記憶域割付け
記憶域解放

レベル3：処理装置管理　高レベル
プロセス創成／抹消
プロセス間メッセージ送信／受信
プロセス停止
プロセス開始

レベル4：装置管理
全入出力装置の状態の把握
入出力のスケジュール
入出力プロセスの起動

レベル5：情報管理
ファイルの作成／抹消
ファイルのオープン／クローズ
ファイルの読込み／書出し

核はプロセスの動作を保持するために必要なルーチンから構成されている．資源管理のどのような機能が別個のプロセスとして遂行されるのであろうか．これを決定するには，どの機能が独立で並列に実行でき，どの機能が順番に実行する必要があるかを知らねばならない．順番に実行する機能を別個のプロセスとしたところで何ら益はない．利用者プロセスの実行と並列に実行できるオペ

レーティング・システムの機能は，ジョブ・スケジューラ，遠隔端末制御および入出力スプーリング・ルーチンなどである．これらを別個のプロセスとすることにする．

各利用者は各装置に対してそれぞれ1つの入出力プロセスを持つこともできる．柔軟なシステムでは，利用者は好きな数だけプロセスを創成することができる．この機能はある種の並列演算や実時間のプロセス制御および時分割の機能のとき有用である．

1.6 OS/MVT の例

IBM の OS/MVT（可変タスク数の多重プログラミング）の大略を説明しよう（IBM，1参照）．各資源については本章の後半で述べるので，本節では詳しく述べない．また，第9章で OS/MVT について詳しく述べるので，本節では OS/MVT をあまり詳しく述べることはせず，これまで述べてきた範囲のことを応用してみよう．

1.6.1 OS/MVT の記憶管理

オペレーティング・システムの中で，記憶管理モジュールは利用者ジョブへの主記憶の割付けおよび記憶域の利用状況の把握を行なう．記憶域は，各ジョブに対して**区画**と呼ばれる一続きのブロックとして割付けられる．図 1.10 においては記憶域はいくつかの部分に分割され，各ジョブは1つの区画を割当てられる．OS/MVT では最大15分割（ジョブ）まで可能である．



図 1.10 OS/MVT の記憶域割付け

1.6.2 OS/MVT の処理装置管理

処理装置管理はジョブに対する処理装置の割付けを行なう．処理装置管理には3つの主要なモジュールがある．それらはジョブ・スケジューラ，処理装置スケジューラおよびトラフィック制御である．

ジョブ・スケジューラは次に開始すべきジョブを選択し，プロセスに変換する．OS/MVT においては幾種類かのスケジューラが使用できる．簡単なものは FCFS（**ファースト・カム・ファースト・サーブド**）の順で実行するものである．他のスケジューラについては第4章および第9章で述べる．

処理装置管理の他の重要なモジュールは**処理装置スケジューラ**である．複数のプロセスが実行状態にされる多重プログラミング・システムにおいては，処理装置スケジューラは処理装置を割当てられるべきプロセスを選択し，処理装置を使用することのできる時間を決定する．OS/MVT の処理装置スケジューラは優先権にしたがって処理装置を割当てられるべきプロセスを決定する．プロセスが処理装置を割当てられると，

1. 入出力を行なうまで，
2. 誤りを犯すまで，
3. 終了するまで，または
4. 優先権の高いプロセスが処理装置を必要とするまで

実行を続ける．OS/MVT には“割当て時間（処理装置は一定の時間だけプロセスに割当てられる）”の選択機能がある．この一定時間が経過すると，処理装置は他のプロセスに割当てられる．

1.6.3 OS/MVT の装置管理

装置管理モジュールはジョブに装置を割当て，これらの装置が効率よく動作するように管理する．ジョブ・スケジューラが1つのジョブを選ぶと，ジョブは入出力装置を要求する．

OS/MVT は，1つの利用者ジョブが他のジョブと共に間歇的に実行される多重プログラミングを行なう．入出力はどのように行なわれるのであろうか．たとえば，もし2つのジョブが1つの印刷機で結果を印刷すれば，オペレータは，各ジョブの結果が混って印刷された紙をはさみで切ったり糊で貼り合せたりしなければならないであろうか．答えは否である．一般に，装置管理が装置を割当てるには3とおりの方法がある．それは**専用**，**共用**，および**仮想装置**として割当てる方法である．印刷機は出力を切ったり貼ったりせずに共用すること

は不可能である．1台の印刷機を1つのジョブに専用させることもできるが，このようにすると装置の効率のよい動作を妨げる．OS/MVT は印刷機のような装置を，スプーリングによって，仮想的に割当てる．スプーリング・モジュールはすべての印刷を仮想的な印刷機，たとえばディスク，に行なう．後刻，スプーリング・モジュールはディスク上に“印刷”された情報を全部取出して最大速度で印刷機に実際に印刷する．

スプーリングは装置管理モジュールのほんの一例にすぎない．ほかにチャネル，制御装置，他の装置などの割当てを行なうモジュールがある．

1.6.4 OS/MVT の情報管理

情報管理はデータ・セット（ファイル）の編成，割付け，保護および検索を行なう．利用者はジョブ制御文を使用してどのファイルが必要となるかを指定する．情報の読出し，書込みおよび保存は，アクセス法モジュールを呼出すことにより，OS/MVT のデータ管理が行なう．利用者のプログラムはファイルを記号名で参照するだけでよく，物理的な記憶動作と記憶域の写像は自動的に行なわれる．

1.7 他の観点より見たオペレーティング・システム

ここでは詳細を述べないが，オペレーティング・システムには他の多くの見方がある．これらを簡単に論じておくことはオペレーティング・システムをよりよく理解するのに役立つ．

1.7.1 歴史的な観点

多くの重要な概念はその真価が認められるよりはるか以前に導入されているので，オペレーティング・システムの発展を正確に述べることは困難である．たとえば，ページングと仮想記憶の概念は最初に 1959 年 Atlas システム (Fotheringham, 1961) で発表された．1960年代の中期には2，3の汎用システムでも使用され，1972年になってはじめて IBM の標準装置の一部として採用することが発表された．本節で述べる歴史的観点および図 1.11 は Saul Rosen (1969) および Robert F. Rosin (1969) の行なった歴史の要約に基づいている．

1.7.1.1 ジョブ単位の処理

初期のシステムではジョブの実行時間がきわめて長く，デバッギングが大変

であったので，プログラマは各自でカードをロードし，ボタンを押し，記憶番地を確認したりして計算機を運転した．

1.7.1.2 エグゼクティヴ・システム

計算機が一層複雑となってくるにつれ，特に入出力管理に関しては，主記憶に常駐して，利用者ジョブに対し入出力制御のサービス（IOCS）を提供するエグゼクティヴ・システムが開発された．実行時間の制限や課金などの他のサービスもこのエグゼクティヴ・システムに加えられた．

1.7.1.3 多重プログラミングのオペレーティング・システム

システムがさらに大きく高速になるにつれ入出力のあい路が増してきた．多重プログラミングは，仕事量の面での効率を上げる技法として採用された．その副産物としてさらに進んだ入出力の標準化，豊富なジョブ制御言語，およびエグゼクティヴ・システムが開発された．

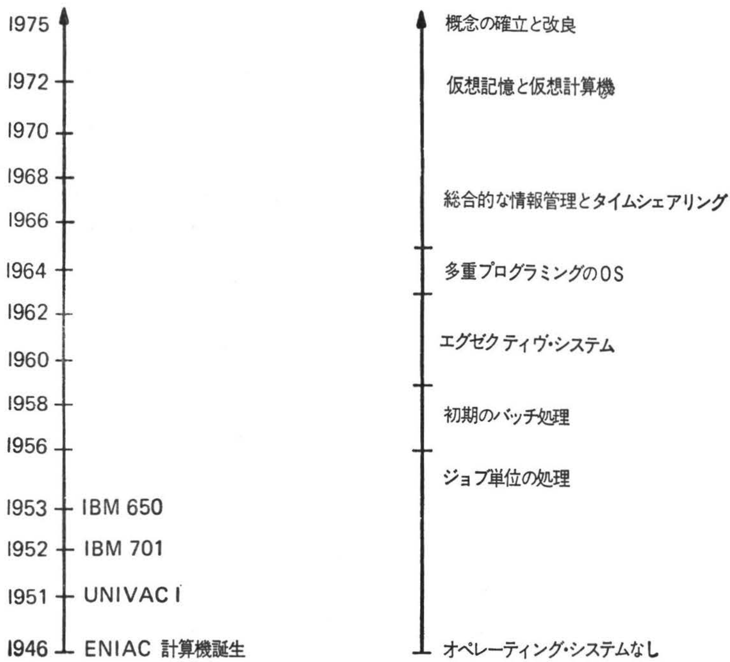


図 1.11 オペレーティング・システムの歴史的な発展

1.7.1.4 総合的な情報管理とタイムシェアリング・システム

単なる数値計算以上の，複雑なデータ処理に対する計算機利用が増すにつれて，データの記憶や処理を標準化し，簡単化することが必要となった．しばしば**データ管理**とか**ファイル・システム**とか呼ばれる進歩した情報管理機能が導入された．同時に時分割（タイムシェアリング）の技法により直接利用者が計算機と会話するという考えが幾つかのシステムに取入れられた．

1.7.1.5 整理統合と改良

急速ではあるが一面無秩序なオペレーティング・システムの発展の中で，多くの重要な考えが出現し，消滅し，また再び別の形で出現したりした．ページングの例はすでに本章の前半で述べた．近年，これまでの発展を整理して改良しようとする努力がなされている．この傾向の一例として，多重プログラミングによるバッチ処理とタイムシェアリングの技法が合併して，バッチとタイムシェアリングの両方の動作ができるオペレーティング・システムができている．

1.7.2 機能的な観点

たいていの利用者は，計算機を特定の問題を解くためのみに使用したいと思う．これらの利用者にとってはページング，セグメンテーション，スプーリングなどといったことは興味のないことである．これらの利用者にとっては問題を定義し，解を求めるための手助けを行なうパッケージが揃っていることのみが必要である．このようなパッケージは重要ではあるが，本書ではこれらをオペレーティング・システムの一部とは考えず，したがって，これらに関して論じない．

これらのパッケージの例としては次のものがある：

1. アセンブリ言語の翻訳プログラム
2. FORTRAN, COBOL および PL/I などのコンパイラ
3. SINE, COSINE および SQUAREROOT などのサブルーチン・ライブラリ
4. サブルーチンを結合しプログラム実行の用意を行なうリンケージ・エディタとプログラム・ローダ
5. SORT/MERGE，テープ複写などのユーティリティ・ルーチン
6. 回路解析やシミュレーションなどの応用パッケージ
7. プログラムのトレーシングや端末への“コア・ダンプ”などのデバッ

ギング・エイド

1.7.3 ジョブ制御言語とスーパバイザ・サービスのインターフェースに関する観点

多くの利用者は，主としてオペレーティング・システムと通信する方法によってオペレーティング・システムを見ている．この通信には通常2つの形がある：

1. 利用者ジョブとシステムへの要求を表わすジョブ制御言語
2. スーパバイザ・サービスのインターフェース，すなわち，利用者プログラムが実行中にオペレーティング・システムのサービスを呼出す方法

複雑で適応範囲の広い最近のオペレーティング・システムに対するジョブ制御言語はしばしば非常に広汎なものである（Brown，1970；Cadow，1970）．利用者はプログラムを実行するための課金と優先順位および必要な資源（入出力）に関する情報を記述できる．典形的なジョブは次のようになる：

註　釈

```
//              JOB     NAME = DONOVAN, ACCOUNT = 6.251,
                        TIMELIMIT = 5, PRIORITY = 8

//STEP1         EXEC    PL1
//OUTPUT        DD      UNIT=TAPE9,VOLUME=SER=
                          0123,DCB=(RECFM=FB,
                          LRECL=80,BLKSIZE=800)
//INPUT         DD      *
           ≡ PL/I プログラム
/*
```

ジョブ・ステップ 1
PL/Iのゴンパイル

```
//STEP2         EXEC    LINKER,COND=(4,LT,STEP1)
//OUTPUT        DD      DSNAME=REAL.LIVE.FILE
//INPUT         DD      DSNAME=&STEP1.OUTPUT
//SYSLIB        DD      DSNAME=PLI.LIBRARY
/*
```

ジョブ・ステップ 2
PL/Iの出力とライブラリとの結合(レベル4以上のコンパイル誤りのないときのみ)

```
//STEP3         EXEC    REAL.LIVE.FILE,MEMORY=100K
//OUTPUT        DD      UNIT=PRINTER
//INPUT         DD      *
           ≡ 入力データ
/*
```

ジョブ・ステップ 3
利用者プログラムの実行

ジョブ制御言語の機能を駆使すると，利用者はジョブ・ステップの動作や実行順序を詳細に指定することができる．たとえば，本節の例では COND=(4, LT, STEP 1) という条件にしたがって，後のステップの実行を制御することができる．このように，ジョブ制御言語は巨視的に見て，強力なプログラム言語である．

利用者に対するスーパバイザ・サービスのインターフェースは，しばしば，特にプログラムが高水準言語によって作成されているときは，陰に隠れたものである．アセンブラ語を使用する場合には，スーパバイザ呼出しやシステム・マクロの命令によってスーパバイザ・サービスを陽に要求することが必要となる．第1.5.1節で示した簡単なプログラムは実際には次のようにプログラムされる：

```
1    MVC    C,B             C=Bと設定
2    SVC    13              80バイトの空き領域を求め，Xに番地を設定
     DC     A(80,X)
3    SVC    01              Xで指定された領域にカードを読込む
     DC     C'CARD'
     DC     A(X)
4    CLC    X(2),=C'/*'     領域Xの最初の2バイトは"/*"か？

5    BE     END             そうならばENDへ行く
```

SVC は IBM 370 シリーズの計算機で使用されるスーパバイザ呼出し命令である．

多くの利用者は，ジョブ制御言語とスーパバイザ呼出しのインターフェースによってオペレーティング・システムを見ている．

1.8 一般的な設計要項

1.8.1 ソフトウェアの設計手順

ソフトウェアの設計は次の手順に基づいて行なわれる：

1. 問題を明確に述べる．
2. 関連するデータ・ベースを列挙する．

3. これらのデータ・ベースの書式を規定する.
4. 解法を考える.
5. 機能単位を求める.
6. 1.から5.を各モジュールについて反復する.

オペレーティング・システムの設計においては，最初に1.から5.までを行なうとシステムに要求される機能に関する報告書が作成できる．手順5.では，オペレーティング・システムは機能的に4つの管理部に分割できることがわかる．本書の議論はこれらの手順を4回反復することによって行なうことができる．すなわち，すべてのモジュールは4種類の資源管理に分割される．2回目の反復によって各資源管理の主要機能（状態把握，決定方針，割付け，および解放）が正確に示される．3回目の反復によってこれらの機能を遂行するためのアルゴリズムを開発することが必要となる．4回目の反復では基本的な表の処理機能，データ・ベースおよび行列処理が認識される．

1.8.2 組込みの道具

最近の大工が単にのこぎりと金鎚と釘だけでは家を建てないように，最近のプログラマは十分な適応性のある道具の助けなしには効率よく仕事を行なうことができない．このような道具を用いると，以前には多数のプログラマが必要であった仕事を一人で行なうことができ，ソフトウェアの道具を効果的に利用すれば仕事をより速く，しかも端麗になしとげることができる．

本節ではシステム設計者や作成者の能率を大幅に改善した2，3の道具——高水準言語，タイムシェアリング・システムおよび構造化されたプログラミング——について述べる．一口にいって，アセンブラ言語をやめて高水準言語を使用するとプログラマの論理の明瞭さと生産性を大幅に増すことができる．強力な情報管理能力を備えたタイムシェアリング・システムを利用すると迅速にしかも容易にデバッグができ，原始プログラムのモジュールの制御と保守が簡単に行なえる．**ソフトウェア工場**という言葉はこのような設備を指して使用されている．拡張計算機に関する節で述べたように，構造化したプログラミングは大きなシステムの概念設計を単純化し，システムの作成とデバッグを容易にするための1つの方法である．

高水準言語はオペレーティング・システムを作成するのにも使用するべきである．それにもかかわらず第7章で例としているオペレーティング・システムはアセンブラ語で書かれている．前に述べたことは誤りなのであろうか．否．

MULTICSを調べてみると95%までは高水準言語（PL/I）で書かれている．アセンブラ語で作られた5%の部分は，最も基礎的で計算機に依存する部分である．第7章の例のオペレーティング・システムは小規模のもので非常に基本的な機能だけから構成されている．

1.8.3 文書化

適切な文書化は必須の事項である．たいていのオペレーティング・システムはきわめて単純なアルゴリズムによっているが，データ・ベースはきわめて複雑である．このようなシステムの文書化を行なう1つの方法は，すべてのモジュールの相互の関係の骨組を示すことである．次に各モジュールについて，そのモジュールがデータ構造に対して何を行なうかの簡潔な説明と共に（たとえばPL/I流に）データ構造を示すことである．第7章の例のオペレーティング・システムの設計の議論では，必ずしも理想的ではないが，適当な文書化のレベルについて十分な説明がなされている．ソフトウェア工場のようにシステム開発のためにタイムシェアリング・システムを利用すると，情報管理機能とテキスト編集機能を使用して，オンラインの最新情報の文書化を管理することが可能となる．

1.9 予告

第2章ではシステムのハードウェアの環境についてその背景を述べる．オペレーティング・システムは資源管理者であるという観点に立って，第3，第4，第5および第6章では各管理（記憶装置，処理装置，装置および情報管理）について論じる．これらの章では各資源に対する種々の管理技法を示す．場合によっては使用されるデータ・ベースと流れ図とを用いて特定のアルゴリズムを述べる．第8章ではこれらの資源管理の相互関係を説明し，例をあげる．第7および第9章では例のオペレーティング・システムについて述べ，第3，第4，第5および第6章で発展された骨組を使用して，実在するオペレーティング・システムを要約してみる．

第10章ではこの分野における多数の重要な仕事に関する文献を註釈をつけて示している．この章では特定の分野に関する文献をすぐ選択できるように相互参照の文献表もつけている．

1.10 要　約

オペレーティング・システムは資源の管理者である．オペレーティング・システムは（たとえば，システムの装置の利用効率を上げるとか，ジョブの仕事量を多くするという）一定の方針と目的にしたがって資源を管理する．

本書は，オペレーティング・システムは資源管理者であるという観点からその骨組を紹介しようとするものである．資源は4種類——記憶装置，処理装置，装置，および情報——ある．資源管理機能はいつ実行されるのか，プロセス状態のモデルを用いるとこれらの機能が実行される時期に関してその大略を述べることができる．資源管理機能はどこで実行されるのか．階層的なモデルを用いるとこれらの機能が実行される場所に関してその大略を述べることができる．

演　習　問　題

1.1　次のものの相違点を述べよ．

1. プロセスと処理装置
2. 利用者，ジョブ，およびプロセス

1.2　多重プログラミングの利点を2つあげよ．

1.3　次のような類似関係を考えることができる．
支配人は CPU と対比できる．
秘書は入出力チャネルと対比できる．
タイプライタは入出力装置と対比できる．

1. 支配人と秘書，タイプライタ相互の通信はどのようにして行なわれるか述べよ（たとえば，支配人は秘書とどうして通信するか）．
2. 各自の仕事のスケジュールはどのようにして行なわれるか．優先権，先取権，および割込みで誰が誰に割込みをかけることができるか（たとえば，誰が支配人に割込みをかけることができるか）などを考慮せよ．
3. 問1および2を CPU，入出力チャネル，入出力装置について答えよ．

1.4　次の命令を特権命令と非特権命令に分類せよ．各命令についてその理由をも述べよ．

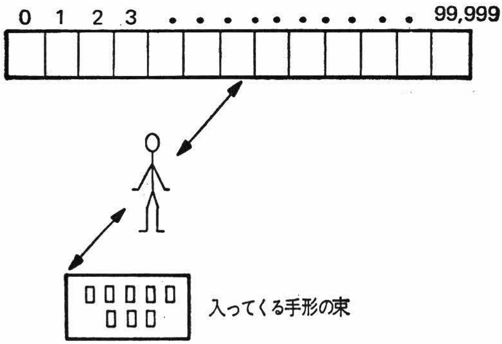
1. テープ駆動装置の動作を開始させる．

2. π の x 乗を求めよ.
3. CPU を停止させる.
4. 時計の値を読む.
5. 記憶装置を帰零する.
6. すべての割込みを不能にする(マスク).
7. 命令番地レジスタの値を変更する.

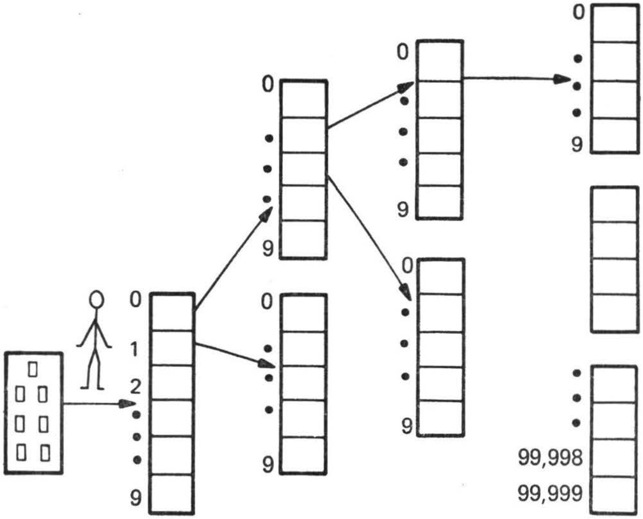
1.5 読者が銀行の小切手支分け部長であるとしよう.この部は99,999人の顧客について一日50,000枚の小切手を処理する.すなわち,50,000枚の小切手は99,999の置き場に分配される.

2とおりの分類法が提案されている.

1. **逐次法**——置き場は1列に並んでいる.置き場を探して小切手をそこに置くのに 99,999/10(約 1,000)秒かかる.



2. **バケット法**——11,110の中間的な置き場を使用する.小切手の第1位の桁によって10に分類し,次に第2位の桁で10に分類する……このような処理には約 16/10＝1秒かかる.



a．すべての小切手を処理するのにそれぞれどれだけの時間かかるか．

b．第2法ではどの段階の処理は（別個のプロセスとして）並列に処理することができるか．すなわち，何人の人を使用できるか．

c．第2法を並列処理で行なった場合にはすべての小切手を処理する時間はどれだけかかるか．

1.6　多数の棚がある倉庫と多数の格納場所のある記憶装置とを対比することができる．多量の品物が入荷し続けるものとする．倉庫の支配人（記憶管理）の仕事はこれらの品物を棚に収納することである．

第1.3節で述べた記憶管理の4つの機能がどのように遂行されるか例を示せ．たとえば，

1. 支配人はどのようにしてスペースを管理するか．
2. 荷卸し場に多数の品物があるときには，支配人はどのようにして品物を収納する順番を決定するのか．
3. 棚はどのように割付けられるか．
4. 1つの品物が棚から取出されたときには支配人は何を行なうか．

1.7　次のような対比によってレストランの経営と処理装置の管理の類似関係をみることができる．

料 理 人——処理装置

料理人頭――処理装置管理

客の注文――プロセス――これらは処理を要求している．たとえば給士を呼んでいる．

客――利用者

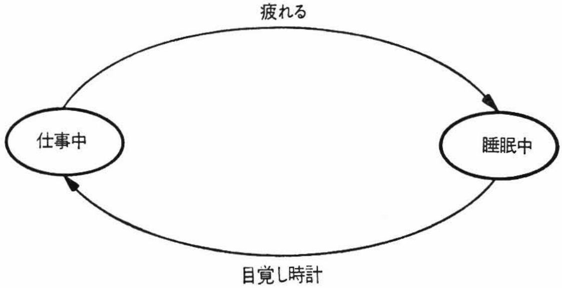
第1.4節で述べた処理装置管理の4つの機能がどのように遂行されるか例を示せ．たとえば，支配人は資源――処理装置，およびプロセスの状況をどのようにして把握するか．

1.8 郵便車が装置で手紙が入出力要求であるとすると郵便局の管理と装置の管理とを対比させることができる．（註：郵便局では手紙がどこへ，どのようにして送られるかの決定は行なわない.）

第1.3.3節で述べた装置管理の4つの機能がどのように行なわれるかの例を示せ．機能2について述べるとき次のことを考慮せよ．手紙の種類が異なれば郵便車を割当てて送り出すか（その手紙は速く到着するが郵便車が不足するであろう），満車になるまで待つか（装置の利用効率は高いが配達が遅い．たとえばマサチューセッツ州のLynnへ行く車が満車になるには1年もかかるであろう）．迅速な応答（手紙を直ちに車で配達する）と利用効率（満車になるまで待つ）との調和について論じよ．

1.9 図書館の司書の役割りと情報管理との対比を行なうことができる．第1.3節で述べた情報管理の4つの機能が図書館においてどのように遂行されるか例を述べよ．

1.10 読者の機能のモデルを図のように示すことができる．

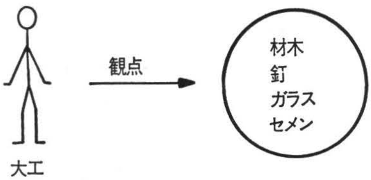


読者に睡眠中と仕事中の2つの状態があり，疲れたときに仕事中から睡眠中へと状態遷移を行なう．

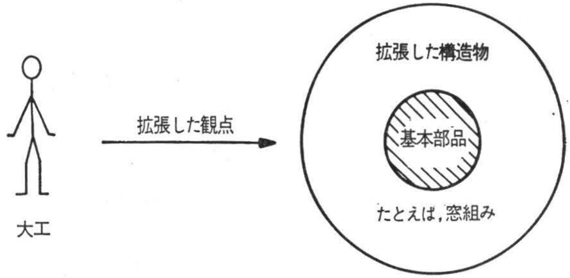
1. 食事中などの状態をさらに3つ追加せよ.
2. 5つの状態の間で起こり得る状態遷移を示せ.

1.11 役に立つ観点の1つに構造化したプログラミング，すなわち階層的な考え方がある（第3の観点は第1.5.2節に示してある）. オペレーティング・システムを構成するために，大工が家を建てるのと対比して，この観点の利点を調べてみよう．大工が使用する基本的な材料は釘，ガラス，セメント，パテおよび木材である.

1. 基礎部品だけを使用する大工はどの点で不利か．たとえば，あらかじめ組立てられているものは何もない.



2. その代わりに，大工は，たとえば窓などの，高度に組立てられた部品を使用することができる．このような部品を列挙してみよ.



このような観点の利点は何か.

3. 2で述べた部品を順に並べてみよ．たとえば，大きな部品が小さな部品を使用していれば前者を後者の上におく（ドアは窓を含む).

1.12 29ページの例題について別のジョブ制御言語を簡潔に定義せよ．この言語は同じ能力を有するものでなくてはならず，たとえばもっと英語に近いといった他の利点を持たねばならない（ファイル入力は STEP 1.

OUTPUT などというように).

1.13 第1.7.1.2節で述べたような多重プログラミングを行なわず，ジョブ（たとえば，コンパイルして，ロードして，実行してほしい原始デックなど）はオペレータに提出されるような原始的なバッチ・システムが与えられたとする．オペレータはジョブのカード・デックをカード読取り機にかける．オペレータは，ジョブごとに必要な資源（たとえばテープや印刷機）とジョブの実行にかかる時間の最大値との情報を与えられている．

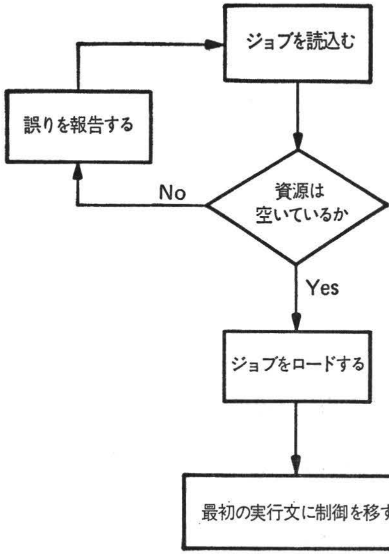
1. オペレータが遂行しなければならない手順をすべて列挙せよ（5～15手順をあげなければならない）．この手順には（どのジョブを先に実行するかを選択するための）決定の動作と（テープをかけたりする）物理的な動作とを含めること．
2. M.I.T. の計算センターの報告によると，ジョブ実行の平均時間は48秒である．読者があげた動作をオペレータが遂行するのにどれだけの時間がかかるか（ジョブを処理する全時間はどれ程か）．
3. どの手順は並列に実行できるか（2人以上のオペレータが居るものとする）．
4. 各手順の処理時間は同一であるとして，並列処理を行なうとジョブを処理するにはどれだけの時間がかかるか．
5. 自動化できる手順はどれか．すなわち，どれがオペレーティング・システムの一部となるか．

1.14 次頁の図はジョブを処理するのに多重プロセシングを行なわない単純なオペレーティング・システムの流れ図である．

1. このオペレーティング・システムに対するジョブ制御言語を作れ．
2. この言語を拡張して，ジョブ・ステップとジョブ・ステップの条件実行を含むようにせよ．
3. 図の流れ図を修飾して2. を処理できるようにせよ．

1.15 問題1. 14では簡単なオペレーテング・システムを設計した．本間では IBM OS/MVT の変更を考えてみよう．

1. 完全な流れ図を書け．
2. 組込みを完成してデバッグせよ．


ジョブを読込む
誤りを報告する
No
資源は
空いているか
Yes
ジョブをロードする
最初の実行文に制御を移す

第 2 章

入出力プログラミング，割込み プログラミングおよび計算機の構造

本章の目的は次のとおりである：

1. 入出力プログラミング，割込み，および非同期処理の概念を説明すること．
2. オペレーテング・システム作成のために理解する必要のある基本的な計算機の構造を説明すること．
3. 後の章の例で使用される IBM 370 の説明をすること．
4. 機械語（アセンブラ語）のプログラミング，入出力プログラミング，および割込みプログラミングの説明をすること．
5. オペレーティング・システムが管理せねばならない重要な資源（処理装置，入出力チャネル，主記憶，入出力装置）を紹介すること．

本章は最近の計算機システム（計算機ハードウェア）を紹介したもので，他の章と内容的に独立している．本章の方針はすべてを包括的に述べるというのではなく，計算機システムに共通の中心的な考え方を読者になれ親しんでもらおうとするものである．読者が例題や演習問題を理解するためにはプログラミング・マニュアルなどの他の資料を参照する必要があろう．

本章では，計算機システムの一般的な構造，入出力プロセスの動作と起動，並列処理の実行と同期，および割込み処理の順に述べる．

本書では，オペレーティング・システムのプログラミングはアセンブラ語[1]で行なえと主張するつもりはないが，計算機システムの構造と能力およびオペ

1) 本章では**機械語**と**アセンブラ語**とは同義語として使用している．問題としているのは機械語（基本的な計算機のハードウェア）であるが，読みやすくするためにアセンブラ語によって記述する．詳細に関しては，Donovan 著・池田訳 “**システム・プログラム**”（日本コンピュータ協会，1974）などを参照してほしい．

レーティング・システムを組込むためのからくりを理解するのにアセンブラ語は都合がよい.

これらの細部に関する知識は，オペレーティング・システムを実際に作成したり研究したりする者にとって不可欠である．したがって，一般的で理論的な枠組を構成する前に，新しい計算機から例を引いてアセンブラ語，入出力プログラミング，および割込み処理について説明する．例は IBM 360 および 370 から選ばれている (IBM 2, 3, 4, 5)[1]. IBM 370 を選んだ理由は次のようなものである：

1. 計算機業界において共通の言語である.
2. 最近の計算機言語の多くと類似している.
3. 機械語，入出力言語，および割込み構造が妥当である.
4. 他の計算機の理解が可能となるようなたとえがそろっている.

他の計算機が大幅に相違する重要な特徴を持っている場合には，本書でもその計算機を紹介する.

本章は 4 つの節に分かれている：

1. 一般的な計算機システムの構造
2. CPU の構造――アセンブラ語のプログラミング
3. 入出力の構造――入出力プログラミング
4. 割込みの構造――割込みプログラミング

読者は未習得の節のみを勉強すればよい．多くの読者にとって第 2.1 節および第 2.2 節は復習のために役立ち，第 2.3 節および第 2.4 節は新しい知識を提供するであろう.

2.1 計算機の構造

図 2.1 は計算機システムの例を示している.

命令やデータは主記憶（伝統的には**コア**とか**メモリ**とか呼ばれている．主記憶，記憶装置，およびコアはしばしば同義語として使用される.）に格納される．何台かの処理装置（命令を解釈するハードウェア装置）が記憶装置に接続される．最近のシステムでは，処理装置は入出力処理装置と中央処理装置に専

1) IBM 360 および 370 は一般に使用されている計算機システムのシリーズである．370 の方が新しいシリーズであり，通常の例にはこちらの方を使用する．たいていの例は両方のシリーズに適用できる.

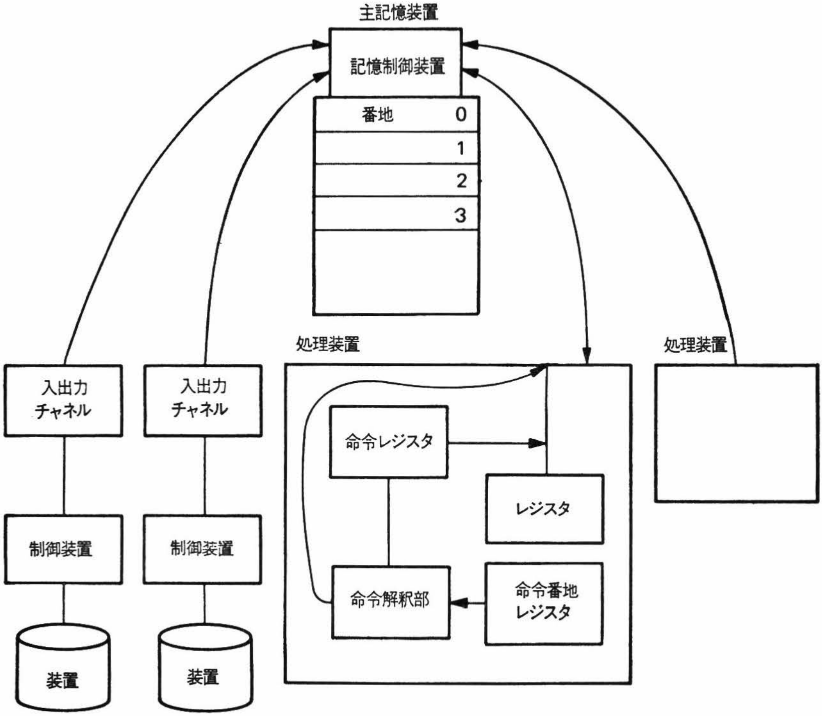


図 2.1　計算機システムの例

用化されることがある．入出力処理装置は，一般に**チャネル**と呼ばれ，主として入力および出力動作に関与する．一方，中央処理装置（CPU）は主として算術演算や論理演算のための命令実行に関与する．1つのシステムに複数台のCPU と入出力処理装置がおかれることもある．

入出力処理装置は，書式も意味も CPU とは異なった命令の組を持つことが多い．第 2.2 節は CPU のプログラミングを，第 2.3 節は入出力処理装置のプログラミングについて述べる．

たいていの CPU は，アキュムレータとして使用される作業用レジスタ，インデックス・レジスタ，およびベース・レジスタによって構成されている．CPU はまた幾つかの内部レジスタ，たとえば命令レジスタや命令番地レジス

タからも構成されている．命令は記憶装置から取出され，命令レジスタに置かれて解釈される．命令番地レジスタは次に実行すべき命令の番地を納めている．

2.2 アセンブラ語のプログラミング

本節では IBM 370 および 370 アセンブラ語を説明する．本節は，主としてアセンブラ語と機械語について幾分かの知識のある人，および IBM 370 の復習をしようとする人を対象として書かれている．アセンブラ語のプログラミングについては多数の入門書があり(Altucher, 1966；Kapur；IBM, 6, 7)，計算機メーカーのマニュアルなども役に立つ．

本節の目的は IBM 370 の詳細を述べることではなく，オペレーティング・システムの機能と機構とを理解するのに必要な特徴のみに限っている．

2.2.1 新しい計算機の一般的な学習法

本節では，新しい計算機を学ぼうとするときに採用するとよい1つの方法を述べる．特に IBM 370 の中央処理装置に関して一連の質問を設け，次にそれらに答える．入出力チャネルは別個の計算機と考えることができるので，同じ質問を入出力処理装置に関しても答える．

1. **記憶装置**

   記憶装置の基本単位，容量および番地指定法はどうなっているか．

2. **レジスタ**

   レジスタは何個あるか．容量，機能および相互の関係はどうなっているか．

3. **データ**

   どんな型のデータが計算機で処理できるか．文字，数，論理データ処理が可能か．データはどのようにして記憶されるか．

4. **命　令**

   命令の種類はどれだけあるか．算術演算命令，論理演算命令，記号処理命令はあるか．書式はどうか．どのようにして記憶されるか．

5. **特殊機能**（オペレーティング・システムに都合のよいもの）

   問1，2，3および4に対する答えによって，たいていの利用者に対しては十分な知識が得られるが，オペレーティング・システムを設計したり理解したりする必要のある利用者はさらに次のようなその他のハード

ウェアの特徴を理解しなければならない．

a．**プログラムの状態**　CPU は，どの命令を実行したらよいのかをどのようにして知るか．CPU はオペレーティング・システム状態と利用者状態とを区別するか．

b．**入出力処理**　入力および出力はどのように処理されるか．別個の入出力処理装置が使用されるときには，CPU はどのようにしてそれと通信するか．

c．**割込み**　どのような種類の割込みが存在するか．割込みはどのように処理されるか．

d．**マスク**　割込みを無視したり保留したりできるか．

e．**保　護**　記憶装置の各部分に対して読出し，書込み，または実行を保護することができるハードウェアが備わっているか．

f．**タイマ**　値を読出したり設定したりできる時計がついているか．

g．**状　態**　CPU は複数の状態をとるか．CPU は特定の状態でのみ実行が許される特権命令を持っているか．

h．**マイクロプログラミング**　利用者は計算機の命令を変更できるか．誰でも可能か．

2.2.2　計算機の構造——IBM 360 と 370

本節では前述の質問を IBM 370 に関して答える．その内容は一般に 360 に対しても適用できる．

2.2.2.1　記 憶 装 置

IBM 370 の記憶装置の基本単位は，8 ビットの情報からなるバイトである．すなわち，記憶装置の各番地は 8 ビットの情報を格納することができる．以下に示すような連続したバイトに対して処理を行なう命令がある：

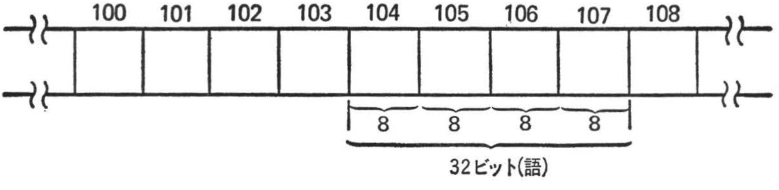
記憶域の単位	バイト数	ビット数
バイト	1	8
半　語	2	16
語	4	32
2語長語	8	64

IBM 370 の記憶装置の容量は 2^{24}（約 1,600 万）バイトである．

IBM 370 における番地指定は 3 つの成分によって行なうことができる．番

地の値は，オフセットの値＋ベース・レジスタの内容＋インデックス・レジスタの内容に等しい．この番地指定の例を後に示す．

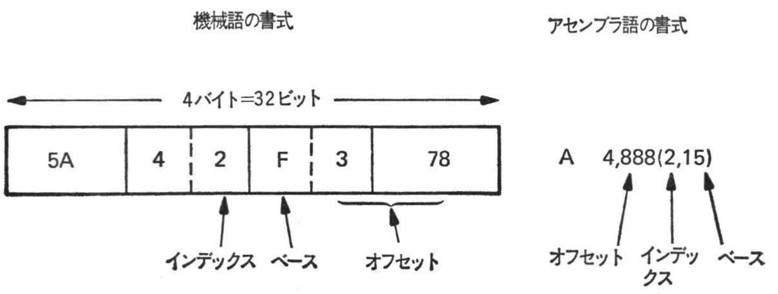
一般に記憶の単位に関する処理は若い方の番地を指定して行なわれる．たとえば，1語（4バイト）を番地指定するときは若い方の番地で，図では104が番地の値となる：



2.2.2.2 レジスタ

IBM 370は16個の汎用レジスタを持ち，それぞれ32ビットから成っている．64ビットから成る浮動小数点レジスタが4個ある．さらに，32ビットの特別な制御レジスタが16個ある．IBM 370は64ビットのプログラム状態語(PSW)を持ち，ロケーション・カウンタの値，保護情報，および割込み情報を格納している：

汎用レジスタは各種の算術演算や論理演算のため，およびベース・レジスタとして使用することができる．プログラマが汎用レジスタを算術演算や論理演算において使用する場合には，これは数が加算されたり，減算されたり，比較されたりなどするメモ用のレジスタであると考えることができる．ベース・レジスタとして使用されるときには，番地の形成に一役買う．次の命令を例として考える：



これは加算命令として解釈される．汎用レジスタ4の内容に1つの数が加算される．その数の番地は 888（オフセット）+レジスタ2の内容（インデックス）+レジスタ15の内容（ベース）である．すなわち，これら3つの数の和がレジスタ4に加算したい数の番地である．

ベースとオフセットによって番地を指定する方法には幾つかの長所がある．この方法では，命令中に絶対番地を用いずに相対番地を使用するので，再配置の処理が容易となる．さらに，番地を指定するために必要となる領域が減少する．IBM 370 の主記憶容量は最大 2^{24} バイトである．もし直接番地指定を行なうならば，番地部に 24 ビット必要である．これに対して，ベース部とオフセット部を合せてただの 16 ビットですんでいる．

2.2.2.3 データ

IBM 370 には，2進整数，2進浮動小数点数，10 進整数，および文字列など数種類のデータを処理する命令がある．典型的なデータの書式は 図 **2.2** に示されている．

すべてのデータと命令は，物理的には2進数の1と0のつながりとして記憶されている．10進数で+300 値をとる 16 ビットの固定小数点数の半語は，2進数'0000 0001 0010 1100'として格納されている．便宜上，2進数は，通常，16 進数（基数 16）によって表わされる．

16 進数の1桁は2進数の4桁と相互に変換可能である．10 進数の+300 は2進数で B '0000 0001 0010 1100' であり，16 進数では
X '0 1 2 C' である．固定小数点の数は半語または1語に格納される（図 **2.2 a** および 図 **2.2 b** 参照)．

IBM 370 では数を10進形式で格納することができる（図 **2.2 c** および **2.2 d** 参照)．すなわち，数は2進数としてではなく，10進数の表現にきわめて近い形で格納される．たとえば，数 12 は1バイトで，上位の4ビットを10進の1（0001）とし，下位の4ビットを10進の2（0010）として格納することができる．10 進形式は事務用のデータ処理には都合がよい．

IBM 370 では図 2.2 に示すように浮動小数点数，論理データ，および文字列を記憶装置の中で表現することができる．

これらの型のデータに対してはそれを処理する命令が用意されている．

2.2.2.4 命令

IBM 370 には算術演算，論理演算，制御の移行，および特殊な入出力命令

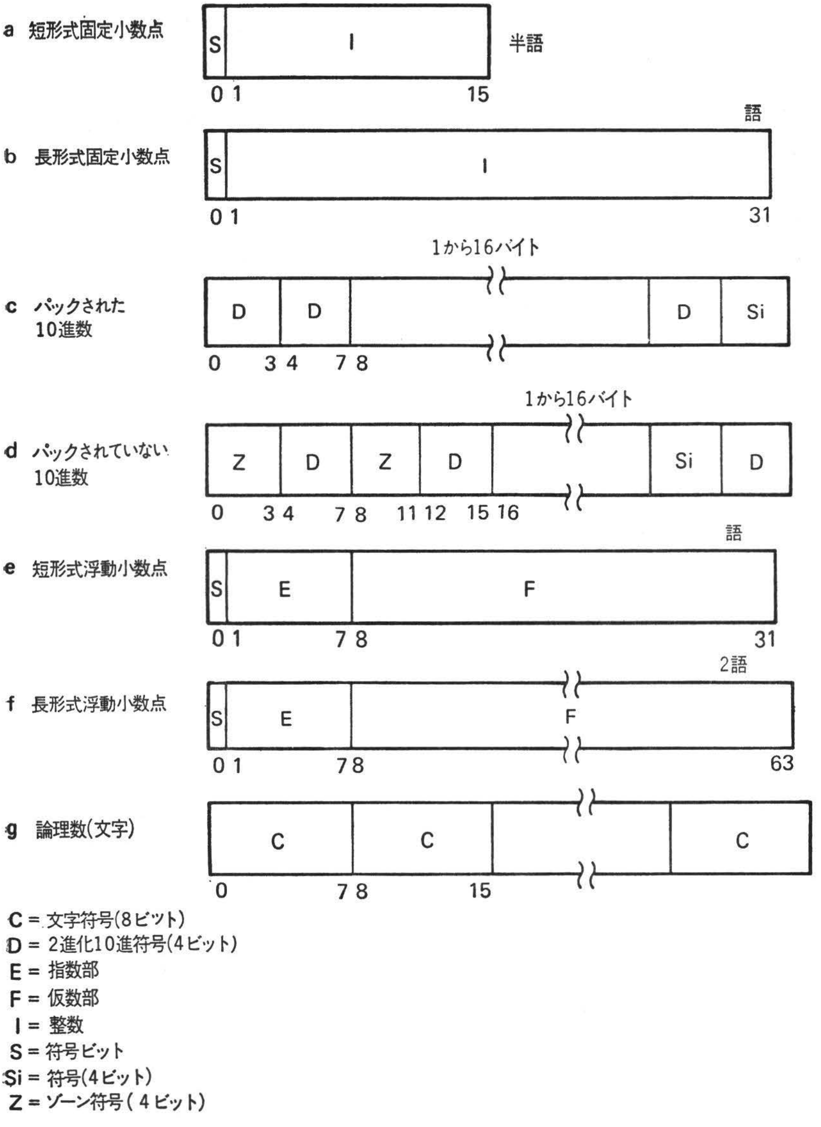


図 2.2 IBM 370 のデータ書式

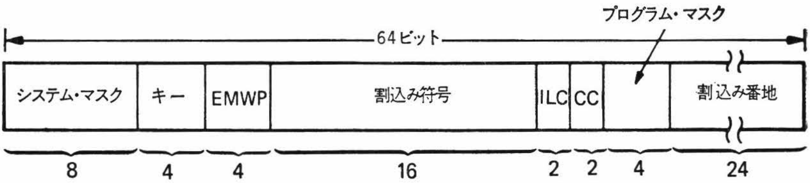
がある．命令には基本的に5種の書式がある．命令の長さは2バイト，4バイトまたは6バイトである．たとえば，語（4バイト）

0101 1010 0011 0010 0101 0000 0001 0000
←――― 4 バイト ―――→

は RX 書式の加算命令として，レジスタ 3 に番地（16＋レジスタ 5 の内容＋レジスタ 2 の内容）の内容を加算すると解釈される．

2.2.2.5 特殊機能

プログラムの状態　プログラム状態語は CPU の状態を規定する内部レジスタである．基本制御モード[1] におけるプログラム状態の書式は図に示すとおりである：



IBM 370 の CPU は次に実行する命令をどのようにして求めるのであろうか．CPU はどのようにして割込み原因を知るのであろうか．CPU は特定の割込みがマスクされていることをどのようにして知るのであろうか．特権状態となっていることをどのようにして知るのであろうか．これらはすべて PSW を通じて行なわれる．

本節では簡単に PSW の各部分について説明する．第 2.4 節では割込み処理においていかにこれらの情報が使用されているかの例を示す．

命令番地は，次に実行すべき命令の番地を納めている．

EMWP 部（E＝拡張モード，M＝マシン・チェック・モード，W＝待合せ状態，P＝プロブレム状態モード）には，本節の議論に関係して W と P の 2 ビットの情報がある．P ビットは CPU のモードを示す（0：スーパバイザ・モード，1：利用者モード）．特定の“特権”命令はスーパバイザ・モードでのみ実行することができる．W ビットは処理装置が実行中（W＝0）または停止中，おそらく割込み待ち（W＝1），であることを示す．

1) IBM 370 には 2 とおりの書式の PSW が設けられている．基本制御モード（E＝0）においては，PSW は IBM 360 と同一である．拡張制御モード（E＝1）においては，制御レジスタと共に特殊機能を遂行する．これについては第 3 章まで述べない．

割込み符号は，最後に起こった割込みの種類を示す符号化された情報である．

システム・マスクは，特定のチャネルからの割込みを CPU が受付けるかどうかを示す．チャネル 0〜5 が CPU に割込みをかけることが認められるときにはそれぞれビット 0〜5 がオンとされる．ビット 6 がオンであればチャネル 6 以上は割込みが認められる．ビット 7 がオンであれば外部割込み（時計，オペレータの緊急割込みボタンなど）が認められる．入出力割込みがマスクされていて割込みが起こったときは，ハードウェアは自動的にその割込みを保留して後刻処理ができるようにする．

プログラム・マスクは前とは別の種類の割込みをマスクするための情報を納めている．この種の割込みがマスクされていて割込みが起こったときは無視される．なお，すべての割込みがマスクできるわけではない．（同様に，読者においても特定の割込みはマスクできないであろう．たとえば，母親の割込みは通常，処理される．）

キーは，保護の目的に使用される．これについては保護に関する節で説明する．**命令長符号**（ILC）は，現在の命令ではなく，最後に実行された命令の長さを納めている．このようにして，プログラマは 1 命令だけ前の命令について調べることができる．**条件符号**（CC）は現在の値が納められている．条件符号は，算術演算や比較演算など特定の命令を実行すると自動的に設定される．

入出力処理 IBM 370 は，他の最近の大規模計算機と同様に，入出力処理の大部分を遂行する別個の処理装置を備えている．この入出力計算機あるいは処理装置は**チャネル**と呼ばれる．CPU は START I/O という命令を実行して入出力を開始させる．START I/O はチャネルに対してチャネル・プログラムの実行を開始させるために行なう通信である．

IBM 370 の CPU には入出力に関する命令が幾つか備えられている．（実際に入出力を行なうチャネルは，読込み，書出し，ページ送りなどもっと多くの命令を持っている．）主要な命令は 3 つある：

START I/O	入出力チャネルにチャネル・プログラムの実行を開始させる．
HALT I/O	チャネルを停止させる．
TEST I/O	装置，制御装置，またはチャネルの状態を調査する．

これらの命令はそれ自体になんら入出力を行なわず，単に CPU が入出力処理装置と通信する手段にすぎない．第 2.3 節において IBM 370 の入出力チャネ

ルの命令について論じる．

割込み　CPU に割込むことはできるか．できる．第2.4節では IBM 370 で使用されている機構を論じ，その例を示す．他の計算機も割込み処理のために似たようなハードウェアを備えている．割込みとは，処理装置に速刻応答させる現象である．（読者が処理装置であるとして）読者がこのページを読んでいるときに友人が訪ねてきたとすると，読者は割込まれたことになる．読者がとる手順は読んでいたページ番号を控え（ハードウェア），次に友人が望んでいることをきき，その要求を満たすべきかを決定する（ソフトウェア）．

IBM 370 には割込み，状態保存，割込み種別の決定，および適当な割込みルーチンを実行する機構が備わっている．

マスク　読者でも割込まれたくないときがある．このような場合には割込みをマスクするであろう．同様にオペレーティング・システムでも前の割込みの処理中には新しい割込みをマスクしておきたい．

IBM 370 の CPU は特定の割込みをマスクすることができる．プログラム状態のビットが設定されているとき，入出力割込みは保留されて残り，他の割込みは無視される．

保　護　IBM 370 の主記憶は 2,048 バイトのブロックに分割されている．各記憶ブロックには 図 2.3 に示したようにロック（4ビットの0～15の数）が対応している．これらのロックは特権命令によって設定したり検査したりすることができる．記憶装置の参照が許される前に，この記憶ブロックに対応したロックとプログラム状態語の保護キーとが比較される．参照はロックとキーとが一致するかまたはプログラム状態語が“マスタ・キー”（0のキー）を持っているときにのみ許される．

IBM 370 には2種のロック——読出し/書込み保護，または書込みのみ保護——がある．書込みのみ保護のロックがかけられているとき（すなわち，“取出し保護”がかかっていない），ロックとキーが一致しなくても読出しは許される．図2.3の例には有効な参照と無効な参照が示されている．もし参照が許されない場合には保護割込みが発生する．

タイマ　IBM 370 には幾つかの時計とタイマが備えられている：

1. TOD タイマ　64ビットの2進のレジスタであり，1マイクロ秒ごとに値が増す．TOD は 1900年1月1日零時グリニッジ標準時に設定してある．TOD は命令で読出したり，設定したりすることができる．
2. クロック・コンパレータ　TOD と共に使用される．クロック・コ

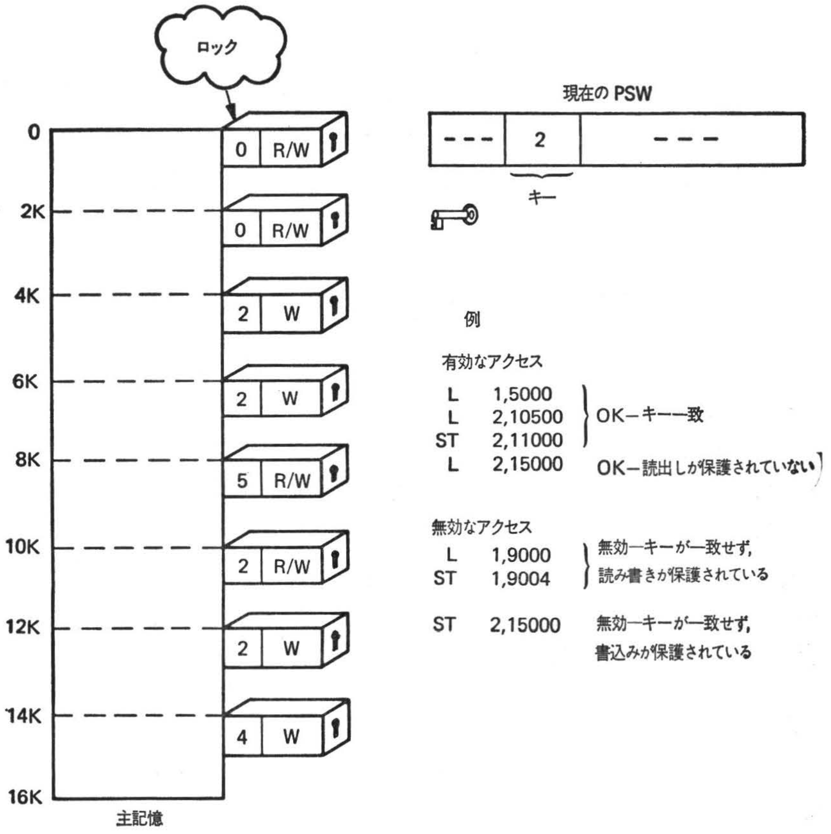


図 2.3 IBM 370 の保護キーの例

ンパレータの値が TOD よりも小さくなる外部割込みが発生する．この機能によって“目覚し時計”として使用することができる．

3. CPU タイマ　特別の 64 ビットのレジスタであり，1 マイクロ秒ごとに値が小さくなる．タイマの値が負になったとき，外部割込みが発生する．これは，CPU タイマに時限を設定し，時間が経ったときに割込みを受取ることにより，眼覚し時計のように使用することができる．

4. インタバル・タイマ　360 との互換性を保つためのものであり，主

記憶の80番地の32ビットの1語である．この語は1/300秒ごとに値が減ぜられる．インタバル・タイマの値が負に変わったとき，外部割込みが発生する．多くの目的のためにはCPUタイマの方がインタバル・タイマよりも正確で目的に適している．

各時計とタイマは単独でも連動してでも使用することができる．

状　態　IBM 370は2つのモード――**スーパバイサ・モード**(PSWのビット15=0)と**プロブレム・モード**(PSWのビット15=1)――のいずれかのモードで動作する．プロブレム・モードは**利用者モード**とか**非特権モード**とか呼ばれるが，このときは“特権命令”を実行することが許されない．利用者がこの種の命令を実行しようとすると，特権処理プログラム割込みが発生する．特権命令には次のようなものがある．

1. プログラム状態語を変更するもの．
2. 記憶保護ロックを変更するもの．
3. 入出力に関するもの．

マイクロプログラミング(Rosen, 1969；Wilkes, 1969)　IBM 370シリーズのたいていのモデルはマイクロプログラミングを行なっている．しかし，利用者がマイクロプログラムを変更できるような機能はない．

2.2.3　機　械　語

本節では機械語（計算機で実際に実行されるコード）について論じる．本節でもIBM 370型計算機の例を使用するが，これらは容易に他の計算機に適用することができる．本節を読んだ後に，読者は自分の使用する計算機の機械語について述べた本かマニュアルを読んでほしい．

機械語を記述するのに1と0の並びや16進数を用いたりせず，覚えやすい記号名の表現を使用する．図2.4は前に第2.2.2.2節で論じられた加算命令を示している．

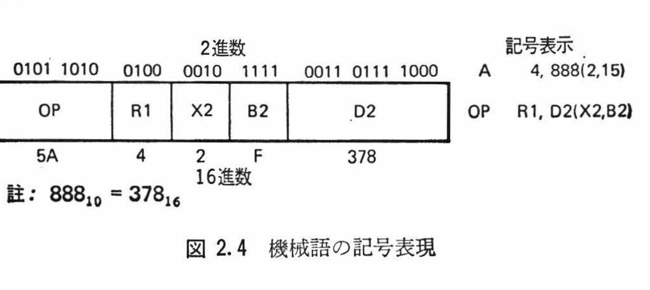


図 2.4　機械語の記号表現

ここで3つの数（21，41，52）を加算する簡単な機械語のプログラムを調べてみよう．このプログラムは 図 2.5 に示されている．例では命令とデータは 1000_{16} 番地からローディングされていると仮定している．

読者は命令の意味を適当なマニュアルを参照して理解してほしい．ここでは結合分岐命令（BALR）と条件分岐命令（BCR）について簡単な註釈を付しておく． BALR 命令は種々の形で使用することができる．例では（第2オペランドが0）次の命令の番地を第2オペランドで指定したレジスタに設定する．BALR 命令は記憶装置の参照を必要とせずに特定の番地の値をレジスタに設定するので，ベース・レジスタの内容の初期化には便利な命令である．なお，例のプログラムではベース・レジスタとしてレジスタ15を使用している．この命令がなぜ使用され，どのように働くかを自ら確かめてほしい．

この小プログラムの実行が完了したとき何をすればよいか． IBM 370 を停止させるのは大変であるし，不経済でもある．したがって，各プログラムはそれを呼出した“親プログラム”に復帰する．呼出し者は利用者のプログラムのこともあるし，オペレーティング・システム自身のこともある．規約により復帰番地は，呼出し者がレジスタ 14 に残している．プログラムが仕事を終えると，レジスタ14に指定された番地に無条件分岐（分岐符号＝15）する．この命令は BCR 15，14 である．

機械語 番地	機械語 内容（16進数）	記号表示 番地	記号表示 内容（10進数）		註　釈
1000	05F0	4096	BALR 15,0		ベース・レジスタ 15＝ 1002_{16}（4098_{10}）
1002	5810F012	4098	L	1,18(0,15)	'21' をロード
1006	5A10F016	4102	A	1,22(0,15)	'41' を加算
100A	5A10F01A	4106	A	1,26(0,15)	'52' を加算
100E	5010F01E	4110	ST	1,30(0,15)	答えを格納
1012	07FE	4114	BCR 15,14		呼出し者に復帰（復帰番地はレジスタ14に入っている）
1014	00000015	4116	21		データ
1018	00000029	4120	41		----
101C	00000034	4124	52		----
1020	----	4128	----		結果

図 2.5　機械語プログラムの例

```
                                          EXTERNAL SYMBOL DICTIONARY
SYMBOL    TYPE ID  ADDR  LENGTH LD ID
EXAM        SD  01 001000 000024

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT    SOURCE STATEMENT

001000                                1 EXAM      START X'1000'        IDENTIFIES NAME OF PROGRAM.
001000 05F0                           2           BALR  15,0           SET REG 15 TO ADDRESS OF
                                      3 *                              NEXT INSTRUCTION.
001002                                4           USING *,15           PSEUDO-OP TELLING ASSEMBLER THAT
                                      5 *                              REG 15 IS BASE REGISTER AND CONTAINS
                                      6 *                              ADDRESS OF NEXT INSTRUCTION.
001002 5810 F012            01014     7           L     1,FIRST        GET '21'.
001006 5A10 F016            01018     8           A     1,SECOND       ADD '41'.
C0100A 5A10 FC1A            0101C     9           A     1,THIRD        ADD '52'.
00100E 5010 F01E            01020    10           ST    1,ANSWER       STORE ANSWER.
001012 07FE                          11           BR    14             RETURN TO CALLER (REG 14
                                     12 *                              SHOULD CONTAIN RETURN ADDRESS
                                     13 *                              AS SET BY BALR 14,15 SUB CALL).
001014 0C000015                      14 FIRST     DC    F'21'
001018 C0000029                      15 SECCND    DC    F'41'
00101C 0C000034                      16 THIRC     DC    F'52'
001020                               17 ANSWER    DS    F              RESERVE SPACE FOR ANSWER.
                                     18           END

                                          CROSS-REFERENCE

SYMBOL.    LEN   VALUE  DEFN      REFERENCES

ANSWER    00004 00102C 00017    C01C
EXAM      00001 001000 C0001
FIRST     C0C04 001014 C0014    0007
SECOND    00004 001C18 C0015    C008
THIRD     C0004 00101C 00016    0009
```

図 2.6　アセンブラ語のプログラム例

2.2.4　アセンブラ語

プログラムを16進の機械語や，それより表現が容易な記号表示の機械語で作ることはきわめて負担が大きく時間のかかることである．アセンブラ語を用いるとより容易にプログラムを作成することができ，プログラマは機械語と同様に命令を直接的に制御することができる．

アセンブラ語の主たる利点は，プログラマが 図 2.6 に示したように記憶番地を記号名で参照できることである．ベース／オフセットによる番地指定への単調な変換はアセンブラによって自動的に行なわれる．アセンブラで自動的にオフセット部を計算するためには，ベース・レジスタとして使用できるレジスタとそのレジスタの内容がわからなければならない．USING 命令はこれら 2 つの点を解決し，アセンブラが正しいコードを生成できるようにする．USING 命令は**擬似命令**である．（擬似命令はアセンブラに対する命令であり，これに対して，機械語は計算機に対する命令である．）定数定義（DC）および記憶域定義（DS）命令は擬似命令であり，アセンブラに数 21，41，および 52 を連続した 3 語に設定させ，さらに答えのために 1 語の領域を確保させる．

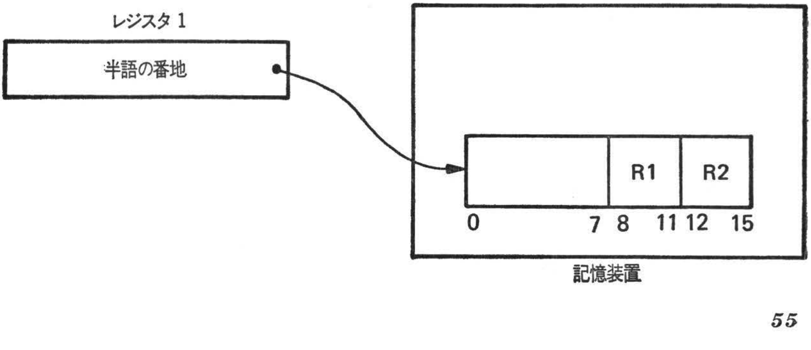
START または CSECT は，アセンブラにプログラムの開始点を示し，利用者がプログラムに名前をつけるための擬似命令である．例ではプログラムの名前は EXAM である．

END はアセンブラにプログラムのカードを終了したことを示す擬似命令である．

2.2.5　アセンブラ語プログラムの例

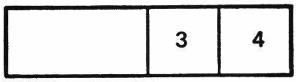
機械語とアセンブラ語の節を完結する前にもっと総合的な例を示そう．この例は後でも使用されるので，よく理解しておいてほしい．

数の整数冪を計算するサブルーチンを作成しよう．このサブルーチンが呼出されるときレジスタ 1 は図に示すような半語を指している：



R1 および R2 はレジスタを指定する．R1 で指定されるレジスタの内容を，R2 の内容の回数だけ累乗する．累乗の結果は R1 で指定されたレジスタに設定される．

たとえば，半語が次のようになっていたとし，レジスタ 3 の内容は10，レジスタ 4 の内容は 2 とすると，サブルーチンはレジスタ 3 に 100 を設定する：



このアルゴリズムは 3 つの場合を処理する必要がある：

指数=0 のとき，値を 1 とする．
指数<0 のとき，値を 0 とする（整数演算を考えている）．
指数>0 のとき，乗算を繰返し値を返却する．

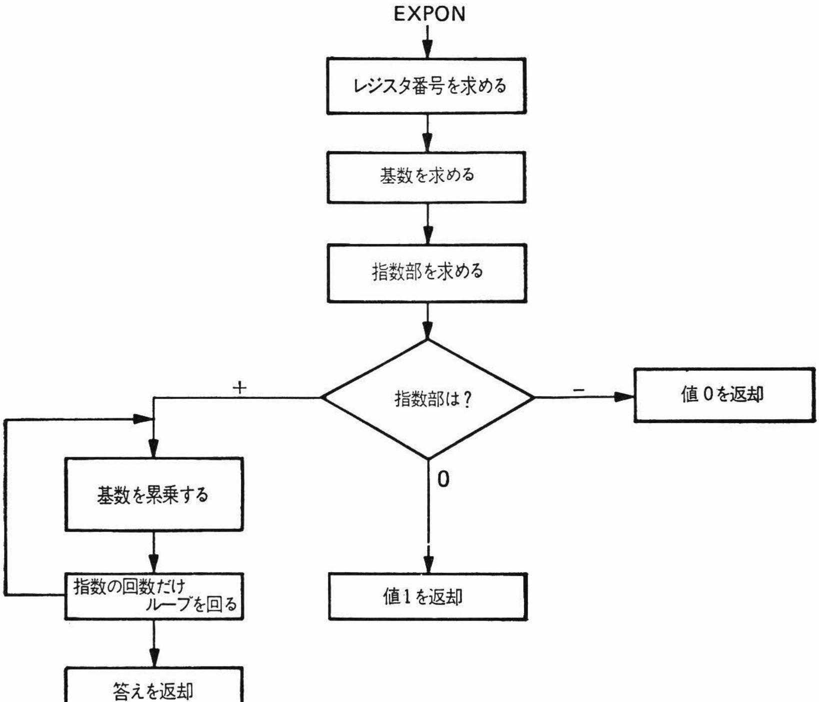


図 2.7　EXPON サブルーチンの流れ図

全体のアルゴリズムは 図 2.7 に示されている.

図 2.8 は EXPON サブルーチンのアセンブラ語のリストである. 左側の第1欄 (LOC) はプログラムの各文の相対番地 (プログラムの先頭からの) である. CSECT および USING 擬似命令はアセンブル・コードを生成せず, 実行時に記憶領域を使用しない. OBJECT CODE の欄は原文に等価な 16 進数表現の機械語を示している. ADDR の欄は各命令の第2オペランドのオフセットの値を示している.

```
 LCC   CBJECT CODE     ADDR1 ADDR2  STMT    SOURCE STATEMENT

CCC000                                 1 EXPCN    CSECT THIS SUBROUTINE IS CALLED BY THE
                                       2 *              SEQUENCE:    L     15,=V(EXPON)
                                       3 *                           BALR  14,15
00C000                                 4          USING EXPON,15        TELL ASSEMBLER REG 15 HAS
                                       5 *                              BASE ADDRESS OF EXPON.
00C000 900F F050             00050     6          STM   0,15,SAVE       SAVE ALL THE REGISTERS.
C00004 1822                            7          SR    2,2             ZERO OUT REGISTER 2.
000006 4320 1001             00001     8          IC    2,1(0,1)        LOAD BYTE 2 OF THE INSTRUCTION.
C0000A 8C20 0004             00004     9          SRDL  2,4             SHIFT EXP REGISTER INTO REG 3.
00000E 8830 001C             C001C    10          SRL   3,28            RIGHT SHIFT EXP REG IN REQ 3.
000002                                11 RBASE    EQU   2               REG 2 CONTAINS BASE REG NO.
C00003                                12 REXP     EQU   3               REG 3 CONTAINS THE EXP REG NO.
C00004                                13 BASE     EQU   4               REG 4 CONTAINS THE BASE.
CCC005                                14 EXP      EQU   5               REG 5 CONTAINS THE EXPONENT.
0C0012 8920 0002             00002    15          SLL   RBASE,2         MULTIPLY BASE REG NO. BY 4.
0C0016 8930 0002             00002    16          SLL   REXP,2          MULTIPLY EXP REG NO. BY 4.
CCC01A 5842 F050             C0050    17          L     BASE,SAVE(RBASE)    FETCH THE BASE.
00001E 5853 F050             C0C50    18          L     EXP,SAVE(REXP)  FETCH THE EXPONENT.
000022 1255                           19          LTR   EXP,EXP         TEST IF EXP NEG, ZERO, POS.
000024 4740 F03C             0003C    20          BM    NEG             TRANSFER IF NEGATIVE EXPONENT.
000028 4780 F042             0C042    21          BZ    ZERO            TRANSFER IF ZERO EXPONENT.
00002C 1874                           22          LR    7,BASE          INITIALIZE RESULT.
00C02E 47F0 F034             00034    23          B     ENCLOOP         MULTIPLY EXP TIMES.
000032 1C64                           24 LOOP     MR    6,BASE          MULTIPLY.
000034 4650 F032             00032    25 ENDLCOP  BCT   EXP,LOOP        LOOP EXP NO. OF TIMES.
0C0038 47F0 F046             CCC46    26          B     DONE
00003C 1877                           27 NEG      SR    7,7             IF NEG EXP, RESULT = 0.
C0003E 47F0 F046             C0C46    28          B     DONE
000042 4170 0001             00001    29 ZERO     LA    7,1             IF ZERO EXP, RESULT = 1.
0C0046 5072 F050             C0050    30 DCNE     ST    7,SAVE(RBASE)   PUT ANSWER IN SAVE AREA REG.
00004A 98CF F050             00050    31          LM    0,15,SAVE       RESTORE THE REGISTERS.
CCC04E 07FE                           32          BR    14              RETURN TO CALLER.
0CC050                                33 SAVE     DS    16F             SPACE TO SAVE THE 16 REGISTERS.
                                      34          END

                                 CROSS-REFERENCE

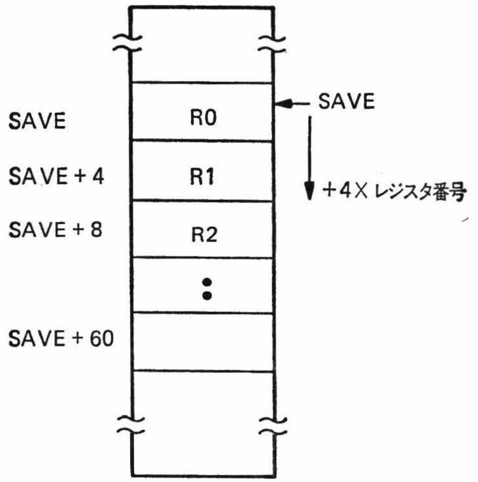
SYMBOL     LEN   VALUE  CEFN     REFERENCES

BASE      00001 000004 00013    0017  0022  0024
DCNE      00004 000046 00030    C026  0028
ENCLOOP   00004 000034 00025    C023
EXP       00001 0000C5 00014    C018  0019  CC19  0025
EXPON     C0001 00C000 00001    C004
LCOP      C0002 000032 00024    C025
NEG       00002 00003C C0027    002C
RBASE     C0001 000002 00011    0015  0017  0030
REXP      00001 0C0CC3 00012    C016  0018
SAVE      00004 000050 00033    0006  0017  0C18  0030  C031
ZERO      00004 000042 00029    C021
```

図 2.8 EXPON のプログラム

アセンブラは，記号，長さ，プログラムの先頭からの相対番地，記号の定義された文番号，および記号が参照された文番号の対応表を作成する．

図 2.8 を理解するうえで 2，3 の註釈を，特に文 15 と 16 について，述べておく，例では SAVE と名づけた領域に，図に示すように，すべてのレジスタの内容を格納している：



各レジスタの内容は SAVE 領域の中で 4 バイトを占めるので，レジスタ RBASE の番地を求めるにはレジスタ番号に 4 を乗じなければならない（この演算は 2 進数で 2 桁左へシフトすることに等しい）.

2.3　入出力プログラミング

初期の計算機システム（たとえば IBM 1401 や 1620）は次の 3 つの基本要素を備えていた：

1. CPU
2. 主記憶装置
3. 入出力装置（周辺装置）

これらの要素は 図 2.9 に示したように接続されていた．

中央処理装置は 図 2.1 に示した一般的な計算機の構成に対応するものであるが，初期のシステムには入出力装置を一時に 1 台ずつ動作させる READ A

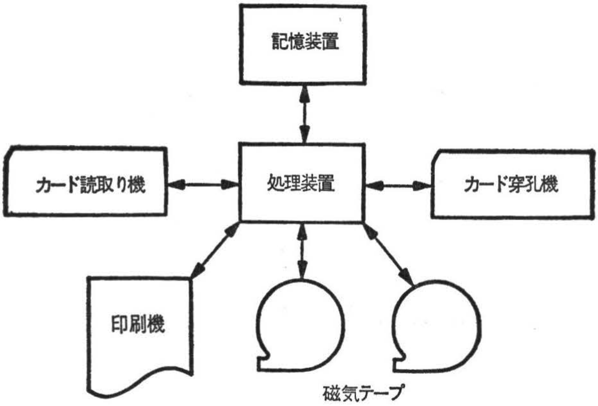


図 2.9 初期の計算機システム

CARD, PUNCH A CARD および PRINT A LINE などの入出力命令も備わっていた．初期の計算機では加算命令は1ミリ秒かかり，READ A CARD 命令は500ミリ秒（最高100枚/分として）かかった．

計算機システムが発展するにつれて CPU 速度と記憶容量が増大し，仕事量が増加した．たとえば，容量1,000バイトでアクセス時間が1ミリ秒の記憶装置が，容量100万バイト以上，アクセス時間1マイクロ秒以下のものに置換わっている．これは速度も容量も1,000倍の増加である．同様に CPU も進歩して典型的な命令実行時間は1マイクロ秒以下になっている．

入出力装置と CPU の記憶装置との間の速度の不釣合によって，**入出力処理装置**（入出力装置と主記憶との間でデータが流れる経路であることから入出力チャネルとも呼ばれる）が開発されることになった．この装置は単純で，特殊目的専用で，しかもあまり高速ではないので，一般には通常の CPU よりも安価である．図 2.10 は各装置ごとに別個の入出力チャネルを備えた計算機システムを示している．すべての入出力がチャネルを経由して行なわれると，CPU はカードを読込むような動作に時間を費やすことなく，高速度で計算を遂行することができる．さらに，複数台のチャネルが同時動作可能であるので，多数のカード読取り機，穿孔機，および印刷機を同時に動作させることができる．しかし，この恩恵を受けるには，オペレーティング・システムがこれらの動作を監督しなければならない．

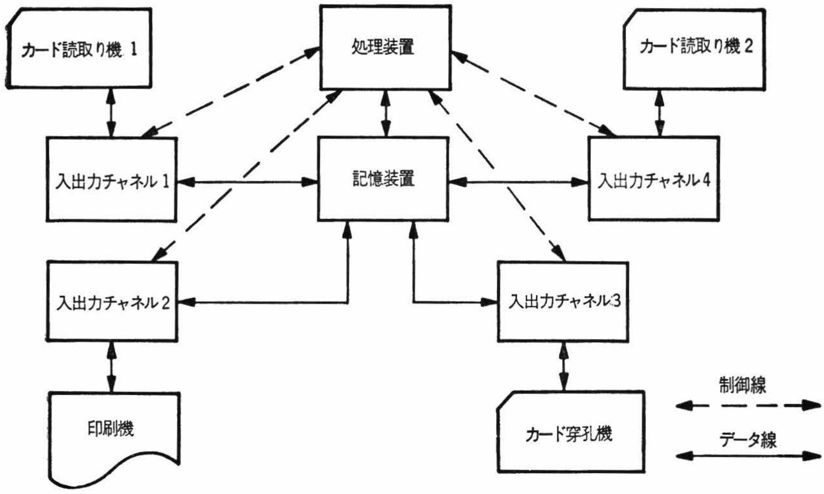


図 2.10　入出力チャネルを持つ計算機システム

2.3.1　入出力チャネルの種類

基本的な原理は同一であるけれども，入出力チャネルには種々の形態と容量のものがあり，きわめて単純な処理装置から高度の CPU に至るものまである．入出力処理装置は安価であるとはいえ，各装置ごとに1台のチャネルを使用することは経済的ではないので，セレクタ・チャネルとマルチプレクサ・チャネルが開発された．

チャネルには複数台の装置が接続できる（IBM 370 ではチャネル1台当たり最大 256 台までの装置を接続できる）．**セレクタ・チャネル**は一時に1台の装置だけにサービスすることができる．すなわち，1台の装置が選択されてサービスを受ける．この種のチャネルは，通常，磁気テープ，ディスク，およびドラムなどの高速入出力装置に対して使用される．この場合入出力要求は即座に完了し，他の装置が入出力を行なうために選択される．

マルチプレクサ・チャネルは同時に多数の装置（最大256台）に対してサービス可能である．このチャネルは，カード読取り機，カード穿孔機および印刷機などの低速の入出力装置に対してサービスを行なうことができる．

ブロック・マルチプレクサ・チャネルは，同一の入出力チャネルで高速の装置に対して，多重にチャネル・プログラムの実行を可能とするような折衷物で

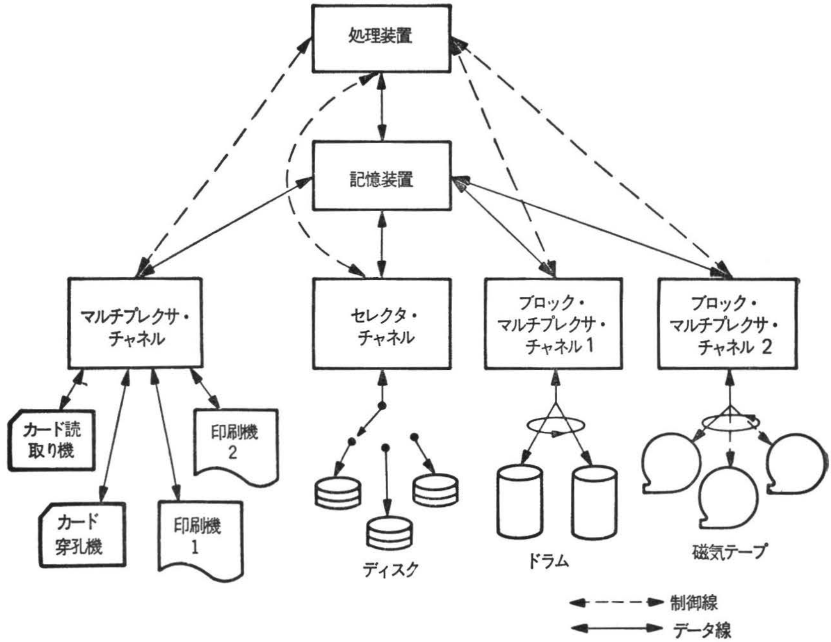


図 2.11 IBM 370 型のシステム構成

ある．ブロック・マルチプレクサ・チャネルは1台の装置に対して1つのチャネル命令を実行すると自動的に他の装置に切換わって1つの命令を実行する．

ブロック・マルチプレクサ・チャネルは，一時に1台の装置のみにサービスする点でセレクタ・チャネルに類似し，すべてのチャネル・プログラムが完了する前に他の装置のサービスができる点でマルチプレクサ・チャネルに類似している．図 2.11 に典型的な IBM 370 の構成を示す．

2.3.2 入出力プログラミングの原理

図 2.11 に示した“輪回しの曲芸”は，相当の工夫がなされないと効率よく働かないことはいうまでもない．曲芸師の役を務め，すべての処理装置を整合するのはオペレーティング・システムの機能である．オペレーティング・システムが処理すべき問題を説明するために，本節では入出力プログラミングに関する基本的な原理を述べる．

中央処理装置は START I/O や HALT I/O などの特定の命令を用いて入出力処理装置と通信する．入出力処理装置は割込み（第2.4節で述べる）によって CPU と通信することができる．

入出力処理装置は独自の命令の組を解釈する．CPU の命令と入出力チャネルの命令を明確に区別するために後者を**入出力コマンド**と呼ぼう．コマンドの組として作られたプログラムのことを**入出力プログラム**といい，このプログラムを作成することを**入出力プログラミング**という．通常，プログラマは自分で入出力プログラムを書くことはせず，入出力プログラムを用意するシステム備えつけの関数を呼出す．モニタ・システムは，利用者の特定の要求に対して入出力プログラムを仕立てて実行を開始させる．

2.3.3 IBM 360 および 370 の入出力処理装置の構造

本節では，第2.2節で中央処理装置に関してとったのと同じ方法で，入出力処理装置を調べてみる．処理装置を調べるときに行なう基本的な5つの質問を出し，それに答え，入出力プログラムの例を示そう．入出力処理装置では，中央処理装置のようには命令が標準化されていない．しかし，その機能の一般化を行なうことは可能である．本節では IBM 370 型の構成から入出力プログラミングの例を示す．

2.3.3.1 記憶装置

記憶装置の基本単位，容量，および番地指定法はどうか．

記憶装置は IBM 370 の中央処理装置が使用すると同一であり，基本単位はバイト，容量は最大 2^{24} である．番地指定に関しては，入出力チャネルは24ビットの絶対番地を使用する．

2.3.3.2 レジスタ

レジスタは何個あるか，容量は，機能は，相互の関係はどうか．

入出力チャネルにはプログラムで指定できるレジスタは備わっていない．しかし，命令番地レジスタおよびデータ・カウンタがある．入出力装置によっては，割算命令などの命令で使用される CPU の内部の作業用レジスタに相当するレジスタを備えていることがある．

2.3.3.3 データ

処理装置で処理できるデータの型は何か．

主として論理型の文字データだけが処理できる（長さが $1\sim2^{16}-1$ の連続したバイト列）．入出力装置によって符号変換の機能を備えていることがある（たとえば，EBCDIC から BCD への符号変換）．入出力処理装置は，入出力装置の状態を示すセンス・データを処理し，この種のデータを読込んだり処理したりすることができる．

2.3.3.4 命　　令

処理装置が解釈するのはどのような命令（コマンド）か．

入出力コマンドは，基本的には3種類に分類される：

1. データ転送：　読込み，逆方向読込み，書出し，センス（装置の状態の読込み）
2. 装置の制御：　制御（改ページ，テープ巻戻しなど）
3. 分　　岐：　チャネル・プログラム内での制御の移行

チャネルは記憶装置からチャネル・コマンド（チャネル・コマンド語（CCW））を取出し，図に示すような書式にしたがって解釈する：

命令符号	データ番地	フラグ	XXXXX（未使用）	回数
0　7	8　31	32　36	37　47	48　63

命令符号（ビット 0〜7）は，実行すべきコマンドを指定する．この符号は，2〜4ビットの命令ビットと4〜6ビットの修飾ビットの2つの部分から成っている．命令ビットは標準的なものであるが，修飾ビットは装置ごとに異なっている．

データ番地（ビット 8〜31）は，コマンドによって参照されるデータの開始番地を指定する．**計数部**はデータのバイト長を指定する．データ番地と計数部は主としてデータの転送を行なうコマンドによって使用される．

フラグは，コマンドをより詳細に規定する．

主要なフラグは次のとおりである：

1. **コマンド・チェイン・フラグ**（ビット 33）は，現在のコマンドが正常に完了したとき，次の CCW を実行することを示す．（**註**：無指定，すなわち，すべてのフラグ=0 のとき，現在実行しているコマンドが完

了するとチャネルは停止する.)

2. **データ・チェイン・フラグ**（ビット 32）は，現在のデータ領域が終了したとき，次の CCW で指定されている記憶域が使用されることを示す.

3. **長さ指定の抑制フラグ**（ビット 34）は，データの長さが指定されたとおりでないとき，プログラムへ通知するのを抑止することを示す（図 2.12 参照).

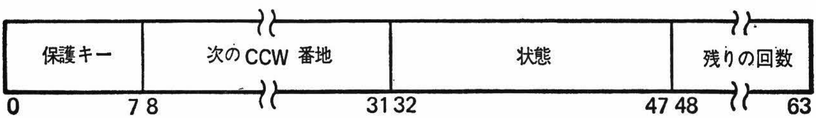
4. **スキップ・フラグ**（ビット 35）は，記憶装置への情報の転送を抑止することを示す.

5. **プログラム制御割込みフラグ**（ビット 36）は，この CCW がチャネルで実行されたとき，チャネルが割込み条件を発生することを示す.

コマンド・チェイン，データ・チェイン，あるいは制御移行で結合された一群の CCW をチャネル・プログラムという. CPU はチャネルを起動してチャネル・プログラムを実行させる.

2.3.3.5 特殊機能

状態　チャネルは，命令番地レジスタの役を果たす内部レジスタを備えている.（マルチプレクサやブロック・マルチプレクサは，装置ごとにこのようなレジスタを備えている.）さらに，状態情報のために記憶装置の特定の 3 語が使用される. 72_{10} 番地から始まるチャネル番地語（CAW）はチャネルで実行される最初の命令の番地を納めている. チャネルが CAW を参照するのは入出力開始命令（SIO）が CPU で実行されている間だけである. チャネル状態語（CSW）は，2 語長語であり，チャネルの状態を示す情報を格納している. 64 番地に置かれているチャネル状態語の書式は次図のとおりである：



キー——チャネルで使用される保護キー

番地——次のチャネル・コマンドの番地

状態——実行中，入出力完了，入出力誤り発生などの表示

回数——チャネル・コマンドで未処理となったバイト数（入出力動作がチ

ャネルで異状終了したとき以外は普通零である)

入出力　チャネル自身が入出力を行なうので該当する項目はない.

割込み　チャネルは CPU に対して割込みを発生することができるが，自分自身が割込まれることはない.

マスク　割込みがないので該当するものはない.

保　護　入出力開始時に保護キー (CAW のビート 0~4) が複写され，入出力データ転送コマンドのすべてに対する保護キーとされる.

マイクロプログラミング　利用者がチャネル・コマンドの命令符号を変更したり，追加したりすることはできない.

2.3.4 入出力チャネル・プログラムの例

新しいページの先頭まで紙を送り，2行印刷し，次のページに改ページするプログラムを書いてみよう．図 2.12 はこの仕事を行なう4つのコマンドを16進数で示している.

例のプログラムは 3211 型印刷機を使用している．第1コマンド (命令符号 8B) は印刷機で用紙をページの先頭まで送る．この命令符号は 3211 型の場合に，チャネルNにスキップする (命令符号 1NNNN001)．したがって，$8B_{16}$ $=1000\ 1011_2$ はチャネル1にスキップすることを示す.

ここでいうチャネルは入出力チャネルのことではない．印刷機には制御用テープまたはチェインがあり，幾つかの列 (チャネル) に穴があいている．規約によって，チャネル1はページの先頭に対応する穴があけられている．これは各種の寸法の用紙 (通常，長さ 11 インチ，13 インチ) が印刷機で使用されることから必要である．第2コマンド，命令符号$=01_{16}$，は 010008 番地に格納された8文字 (GOOD DAY) を印刷する．次のコマンド，命令符号$=0B_{16}$，

番地	命令符号	データ番地	フラグ	未使用	回数	
400	8B	XXXXXX	40	00	XXXX	次ページに送る
408	01	010008	60	00	0008	行の印刷
410	0B	XXXXXX	40	00	XXXX	1行送り
418	89	010010	00	00	0014	20文字の行を印刷，次ページへ送って停止

010008　'GOOD DAY'
010010　'YOU ARE NOW ON A 370'

図 2.12 入出力チャネル・プログラムの例

は用紙を1行送る．前の命令符号 01 は用紙を送らないのでこのコマンドが必要である．このコマンドがないと3つの 01 コマンドによって同じ行に印刷が行なわれる——紙の節約にはなるが結果を読むことが困難である．第2および第3コマンドは，1行印刷後1行紙送りする1つのコマンド，命令符号$=09_{16}$，を用いることもできる．最後のコマンド，命令符号$=89_{16}$，は 010010_{16} 番地に格納された 20 文字（YOU ARE NOW ON A 370）を印刷した後に用紙を次のページの先頭に送る．

図 2.12 では，最後のコマンド以外はコマンド・チェイン・フラグが設定されている(40_8)．加えて，3211 型印刷機で使用される標準のデータ長は 132 バイトであり，8バイトと一致しないので，長さ指定の抑制（SLI）フラグ(20_8)が必要である．SLI フラグが設定されている場合には，第2コマンドを実行した後でプログラムが停止するであろう．SLI は最後のコマンドでも必要となるが，このコマンドを実行して入出力プログラムが停止するので，実際には不要である．データ番地部と計数部は，1行送りや改ページなどの制御コマンドでは使用されない．

図 2.12 はチャネル・コマンドの 16 進表現を示す．この表現にアセンブラ語に対応するものがないのであろうか．“その種”のものはある．IBM 370 のアセンブラ語には入出力プログラミングを容易にする擬似命令 CCW が用意されている．図 2.12 をアセンブラ語で示したものが 図 2.13 である．

CCW は4つの部分——命令符号，番地，フラグ，および計数部——が“，”で区切られている．アセンブラは，DATA1 などのデータの番地を正しく自動的に決定して，CCW の番地部に入れる．

```
              :
              CCW     X'8B',*,X'40',1          次ページに送る
              CCW     X'01',DATA1,X'60',8      行の印刷
              CCW     X'0B',*,X'40',1          1行送り
              CCW     X'89',DATA2,0,20         20文字の行を印刷,
                                               次ページへ送って停止
              :
DATA1         DC      C'GOOD DAY'
DATA2         DC      C'YOU ARE NOW ON A 370'
              :
```

図 2.13　アセンブラ語の入出力プログラムの例

```
         :
         CCW    EJECT,*,CONTINUE,1              改ページ
         CCW    PRINT,DATA1,CONTINUE+SLI,8      行の印刷
         CCW    ADVANCE1,*,CONTINUE,1           1行送り
         CCW    PRINTEJT,DATA2,STOP,20          20文字の行を印刷,
                                                次ページへ送って停止
         :
DATA1    DC     C'GOOD DAY'
DATA2    DC     C'YOU ARE NOW ON A 370'
         :
EJECT    EQU    X'8B'          改ページの命令符号
PRINT    EQU    X'01'          1行印刷の命令符号
ADVANCE1 EQU    X'0B'          1行送りの命令符号
PRINTEJT EQU    X'89'          1行印刷後改ページの命令符号

CONTINUE EQU    X'40'          コマンド・チェインのフラグ
SLI      EQU    X'20'          長さ指定の抑制フラグ
STOP     EQU    X'00'          チャネル・プログラム停止のフラグ
         :
```

図 2.14　記号表現によるアセンブラ語の入出力プログラム

図 2.13 の入出力プログラムは図 2.12 に示した 16 進表現のものよりわかりやすいが，それでもまだ読みづらい．しかし，少し工夫すると図 2.14 に示したようにずっとわかりやすい記号表現のアセンブラ語とすることができる．

2.3.5 CPU とチャネルの通信

チャネル自体がどのようにして動作するかを調べたので，次にチャネルがどのようにして CPU と通信するかを調べよう．チャネルを設けた目的は，入出力動作の細部まで CPU が制御しなくてもよいようにするためであった．CPU とチャネルは通常は主人と奴隷の関係にある．すなわち，CPU はチャネルに動作開始，停止，あるいは動作の変更を命令する．これに対して，チャネルは通常 CPU から指定されるまではどんな動作も開始することができない．

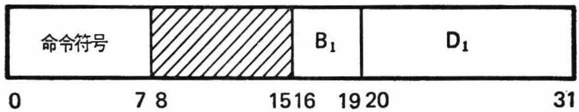
CPU とチャネルの通信は 2 とおりある：

1. CPU からチャネル入出力命令は CPU によって起動される．

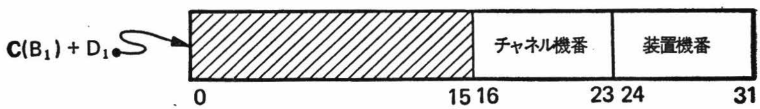
2. チャネルから　　CPU 割込みはチャネルによって起こされる．

本節では最初の方——入出力命令とチャネルとの関係——について述べる．後の方の通信——入出力割込み——については第2.4節にて述べる．

CPU の入出力命令はすべて図のような修飾された SI 形式をとる：



チャネルと装置の番号はレジスタ B1の内容と D1部の内容との和で指定される．和のビット 16～23 はチャネル番号であり，ビット 24～31 はそのチャネルに接続された装置番号である：



次の3つの入出力命令について調べる：

1. **入出力開始** (SIO)：入出力を開始させるには2つの項目が必要である．
   1　チャネルと装置の番号
   2　チャネル・プログラムの先頭番地

   SIO X'00E' という入出力開始命令は，チャネル番号が0で装置番号が 0E であることを指定する．記憶装置の 72～75 番地には CAW が格納されており，チャネル・プログラムの先頭を指定する．
2. **入出力検査** (TIO)：CPU は，指定のチャネルと装置の状態（動作中か否か）を条件符号に設定して表示する．条件符号は標準の条件分岐命令によって検査することができる．
3. **入出力停止** (HIO)：指定の入出力装置とチャネルの動作を直ちに停止する．

SIO または TIO 命令を実行後，CPU には次の条件符号が設定される：

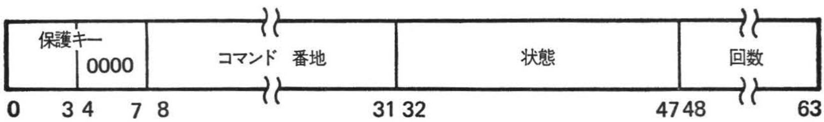
8　正常（動作中でない）．

2 動作中.

1 運転できない.

4 詳細が CSW に設定されている. CSW は 64 番地に格納されている.

チャネル状態語 (CSW) は, 入出力装置の詳細な状態や入出力動作が完了した状態を示す. CSW は, 入出力割込みの過程や, 場合によっては, 入出力開始, 入出力停止, および入出力検査命令の実行中に設定される. CSW の書式は次図のとおりである:



SIO 命令は, チャネルが条件符号 8 を返却したときに限り入出力を起動する. 他の条件符号が返却されたときには, チャネルは入出力要求を却下する. 拒否された理由は条件符号または CSW から判明する.

入出力割込み機構は, 第 2.4 節に説明するように, 有力なものであるが, 単純な入出力処理を遂行するのには必要でない. たとえば, 入出力割込みが無効にされていたとしても, 次の命令列は入出力プログラムを起動し, その動作が正常に完了したことを判定することができる:

```
            :
            :
        LA     1,CCWADDR        入出力プログラムの設定
        ST     1,72             CAWに番地を設定
        SIO    X'00E'           X'00E'の装置の入出力を起動
        BC     4+2+1,ERROR      CC=8でなければ誤り
TESTIO  TIO    X'00E'           入出力が完了したかを検査
        BC     4+2,TESTIO       まだ動作中のときは再び検査
            :
            :
```

入出力検査を行なう度に条件符号は次のようになっているであろう：

TIOの条件符号

2～動作中
2～動作中
⋮
2～動作中　　入出力処理中
4～CSWが格納されている　　入出力完了
8～OK(動作中でない)
CC≠4または2のときループを終了する

TIO 命令の条件符号が2となる回数はきわめて大きいであろう．たとえば，1行印刷するのに約60ミリ秒かかる（1,000行/分として）．TIO 命令と条件分岐（BC）命令がそれぞれ1マイクロ秒かかるとすると，条件符号＝2（動作中）で 30,000 回ループを回るであろう．これが割込み機構が使用される理由の1つである．入出力が完了すると**割込み保留**条件が発生する．すなわち，チャネルは CPU に入出力が完了したことを報告しようとするが，入出力割込みがマスクされている．装置はもう動作中ではないので，次に TIO 命令を実行すると条件符号は2でなく，4となっている．これはチャネル状態語の情報が64～71番地に格納されていることを示している．

この時点でプログラムはCSWの状態部を調べて，本当に入出力が正常に完了したかを判定することができる．チャネルは CPU にすべての情報を与えたので，割込み保留の条件は自動的に消去される（割込みを抹消する）．次に TIO 命令を実行すると，装置は動作中ではないし，割込みも保留されていないので，すべてが終了したことを示す条件符号8を返却する．

2.3.6 単一バッファリングによる入出力の例

例題として一連のカードを読込み印刷してみよう．1つの単純な方法の流れ図を 図 2.15 に，コーディングを 図 2.16 に示す．

この例題プログラムでは，CPU が働いているときはチャネルが遊び，チャネルが働いているときは CPU が（TIO 命令を実行しながら）遊んでチャネルの動作完了を待合せるので，入出力チャネルの利点を活用していない．さらに，カード読取り機（装置 X′00C′）の速度が毎分 1,000 枚，印刷機（装置 X′00E′）の速度が毎分1,000行とすると，このプログラムは毎分500枚の割

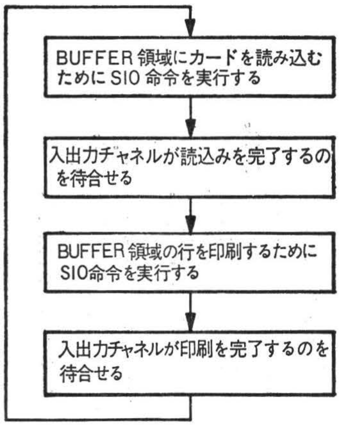


図 2.15 カードを読込んで印刷する単一バッファリングのプログラムの流れ図

りでしか処理（読込みおよび印刷）できない．すなわち，実際には読取り機が動作しているときは印刷機が遊び，印刷機が動作しているときは読取り機が遊ぶことになる．最後に，このプログラムは，入出力誤り処理や，完了処理といった実際に必要とされる他の処理については何の配慮も行なっていない．（**注：**プログラムは永久に動作し続けようとする．）本方式はきわめて単純であることが唯一のとりえである．とにかくこのプログラムは一応動作する．

2.3.7 ダブル・バッファリングによる入出力の例

前の例の処理速度を毎分 500 枚から 1,000 枚に改善することができるであろうか．**ダブル・バッファリング**と名づけられた技法によってこれが可能である．以前には，バッファの内容を印刷し終わるまで1つのバッファに次のカードを読込めないことが問題であった．ダブル・バッファリングを行なうと，バッファ領域1にカードを読込み，この領域の内容を印刷している間にバッファ領域2に次のカードを読込む．バッファ領域1の印刷を終了すると，このバッファ領域1に読込みを開始することができる．この処理を繰返す．図 2.17 に方式を示し，図 2.18 にダブル・バッファリングを行なうプログラムを示す．

2.3.8 複数レコードのバッファリング

前の2つの例では簡単な1コマンドの入出力プログラムを4つ，および2つ

```
                                        EXTERNAL SYMBOL DICTIONARY
SYMBOL    TYPE ID  ADDR   LENGTH LD ID
LIST       SD  01 001000 000099

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT
001000                                1 LIST     START X'1000'
001000 05F0                           2          BALR  15,0           ESTABLISH BASE REGISTER.
001002                                3          USING *,15           TELL THE ASSEMBLER.
001002 8000 F096      01098           4          SSM   =X'00'         TURN ALL I/O INTERRUPTS OFF.
                                      5 *
001006 4110 F036            01038     6 LOOP     LA    1,READ         SET CAW TO CHANNEL
00100A 5010 0048            00048     7          ST    1,CAW            PROGRAM FOR READING.
00100E 9C00 000C      0000C           8          SIO   X'00C'         START READER.
001012 9D00 000C      0000C           9          TIO   X'00C'         WAIT UNTIL READER
001016 4770 F010            01012    10          BNZ   *-4               IS FINISHED.
                                     11 *
00101A 4110 F03E            01040    12          LA    1,PRINT        SET CAW TO CHANNEL
00101E 5010 0048            00048    13          ST    1,CAW             PROGRAM FOR PRINTING.
001022 9C00 000E      0000E          14          SIO   X'00E'         START PRINTER.
001026 9D00 000E      0000E          15          TIO   X'00E'         WAIT UNTIL PRINTER
00102A 4770 F024            01026    16          BNZ   *-4              IS FINISHED.
00102E 47F0 F004            01006    17          B     LOOP           PROCESS NEXT CARD.
                                     18 *
001032 000000000000
001038 0200104800000050              19 READ     CCW   X'02',BUFFER,X'00',80    READ A CARD.
001040 0900104800000050              20 PRINT    CCW   X'09',BUFFER,X'00',80    PRINT A CARD.
001048                               21 BUFFER   DS    CL80           80-BYTE BUFFER AREA.
                                     22 *
000048                               23 CAW      EQU   72             CAW = LOCATION 72.
                                     24          END
001098 00                            25                =X'00'

                                          RELOCATION DICTIONARY
 POS.ID    REL.ID    FLAGS    ADDRESS
   01        01        08     001039
   01        01        08     001041

                                       CROSS-REFERENCE
SYMBOL    LEN   VALUE  DEFN       REFERENCES
BUFFER   00080 001048 00021     0019  0020
CAW      00001 000048 00023     0007  0013
LIST     00001 001000 00001
LOOP     00004 001006 00006     0017
PRINT    00008 001040 00020     0012
READ     00008 001038 00019     0006
```

（STMT 1～17：CPUにより実行）
（STMT 19～25：チャネルにより実行）

図 2.16 カードを読込んで印刷する単一バッファリングのプログラムのリスト

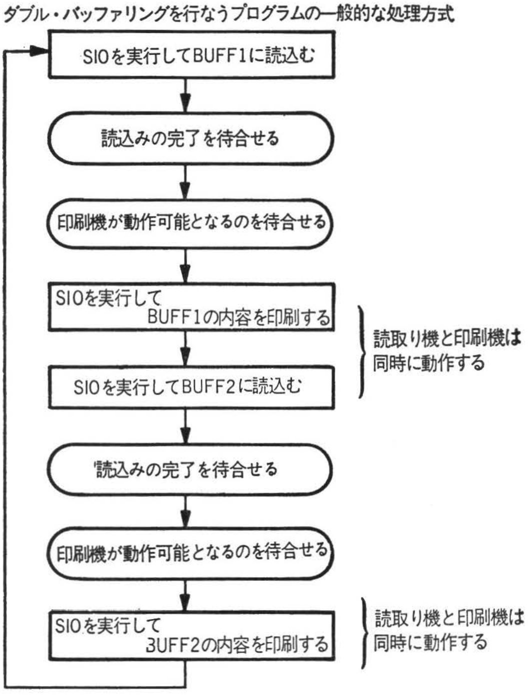


図 2.17 ダブル・バッファリングによってカードを読込み印刷する処理

```
  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT

CC1000                                  1 LIST     START X'1000'
001000 05F0                             2          BALR  15,0              ESTABLISH BASE REGISTER.
001002                                  3          USING *,15              TELL THE ASSEMBLER.
001002 8000 F11E       01120            4          SSM   =X'00'            TURN ALL I/O INTERRUPTS OFF.
                                        5 *
001006 4110 F05E             01060      6 LOOP     LA    1,READ1           SET CAW TO READ1 CHANNEL
00100A 5010 0048             00048      7          ST    1,CAW                PROGRAM (BUFFER1).
00100E 9C00 000C       0000C            8          SIO   X'00C'            START READER.
CC1012 9D00 000C       0000C            9          TIO   X'00C'            WAIT UNTIL READER
001016 4770 F010             01012     10          BNZ   *-4                  IS FINISHED.
                                       11 *
00101A 4110 FC6E             01070     12          LA    1,PRINT1          SET CAW TO PRINT1 CHANNEL
00101E 5010 0C48             CCC48     13          ST    1,CAW                PROGRAM (BUFFER1).
001022 9D00 000E       0000E           14          TIO   X'00E'            MAKE SURE THAT THE PRINTER
001026 4770 F020             01C22     15          BNZ   *-4                  IS NOT BUSY.
00102A 9C00 000E       0000E           16          SIO   X'COE'            START PRINTER.
                                       17 *
00102E 4110 F066             01068     18          LA    1,READ2           SET CAW TO READ2 CHANNEL
001032 5010 0048             00048     19          ST    1,CAW                PROGRAM (BUFFER2).
001036 9C00 C00C       0000C           20          SIO   X'00C'            START READER.
00103A 9D00 000C       0000C           21          TIC   X'00C'            WAIT UNTIL READER
0C103E 4770 F038             0103A     22          BNZ   *-4                  IS FINISHED.
                                       23 *
001042 4110 F076             C1078     24          LA    1,PRINT2          SET CAW TO PRINT2 CHANNEL
001046 5010 0048             00048     25          ST    1,CAW                PROGRAM (BUFFER2).
CC104A 9D00 000E       0000E           26          TIO   X'00E'            MAKE SURE THAT THE PRINTER
00104E 4770 F048             0104A     27          BNZ   *-4                  IS NOT BUSY.
C01C52 9C00 0C0E       0000E           28          SIO   X'00E'            START PRINTER.
                                       29 *
001056 47F0 F004             01006     30          B     LOOP              PROCESS NEXT CARD.
                                       31 *
00105A 0C0000C00000
001060 0200108000000050                32 READ1    CCW   X'02',BUFFER1,X'00',80    READ CARD INTO BUFFER1.
001068 02C010D00CCC0050                33 READ2    CCW   X'02',BUFFER2,X'00',80    READ CARD INTO BUFFER2.
001070 0900108000000050                34 PRINT1   CCW   X'09',BUFFER1,X'00',80    PRINT CARD FRCM BUFFER1.
001078 090010D000000050                35 PRINT2   CCW   X'09',BUFFER2,X'00',80    PRINT CARD FROM BUFFER2.
001080                                 36 BUFFER1  DS    CL80              80-BYTE BUFFER AREA.
0010D0                                 37 BUFFER2  DS    CL80              80-BYTE BUFFER AREA.
                                       38 *
000048                                 39 CAW      EQU   72                CAW = LOCATION 72
                                       40          END
001120 00                              41                =X'00'
```

図 2.18　ダブル・バッファリングによってカードを読込み印刷するプログラム

のカード1枚分のバッファを使用した．1ページにちょうど60行印刷しようとすれば，CPU プログラムを変更する必要が生じる：

1 カード60枚分のバッファを使用する：

```
BUFFER1    DS    60CL80    60枚のカードのためのバッファ
BUFFER2    DS    60CL80    60枚のカードのためのバッファ
```

2 入出力プログラムを次図のように変更する：

```
READ1       ┌CCW    X'02',BUFFER1,X'40',80
            │CCW    X'02',BUFFER1+80,X'40' 80
60個のCCW   │ :
            │ :
            └CCW    X'02',BUFFER1+59*80,X'00',80

PRINT1      ┌CCW    X'8B',*,X'40',1
            │CCW    X'09',BUFFER1,X'60',80
61個のCCW   │CCW    X'09',BUFFER1+80,X'60',80
            │ :
            │ :
            └CCW    X'09',BUFFER1+59*80,X'00',80
```

このように変更すると一時に60枚のカードを読込み，新しいページに印刷する．この変更によって効率や速度が著しく向上するわけではないが，後で割込み処理において説明する他の利益が得られる．

2.4 割込みの構造と処理

前に簡単に割込みについて述べた．割込み機構を組込んだり定義するには多数の方法がある．本節では次のような定義を行なう：

割込みとは，

1. 非同期的なあるいは例外的な事象に対する応答で，
2. 後刻再起動できるように現在の CPU 状態を自動的に保存し，
3. 指定されたルーチン（**割込みハンドラ**と呼ばれる）に自動的に制御を移す

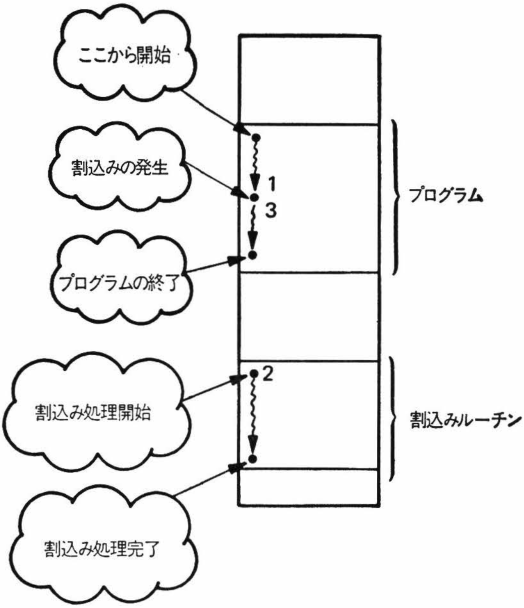


図 2.19　割込みにおけるプロセスの軌跡

ことをいう．

例として，図 2.19 に割込まれた処理装置の動作の径路を示している．点1に至るまで処理装置はプログラムを実行していた．ここで割込みが起こり，処理装置の制御を割込みルーチン2に強制的に移した．割込みルーチンの終了時に処理装置は点3から元のプログラムの処理を再開した．

2.4.1　割込みの種類

本節では割込みを定義し，IBM 370 の機能と関係づけてみよう．
(割込み処理の詳細は本書の範囲を越える．本書では割込み条件の処理に関する枠組を述べる．)

IBM 370 では，種々の割込みは5種類に分類されている． 各種の例を次に示す：

1.　入出力割込み

a 不正入出力コマンド
b 入出力チャネル動作終了（チャネル・エンド）
c 入出力装置動作終了（デバイス・エンド）

2. プログラム割込み
a 不正 CPU 命令
b 固定小数点演算あふれ
c 記憶域侵害

3. スーパバイザ呼出し割込み（SVC 命令）

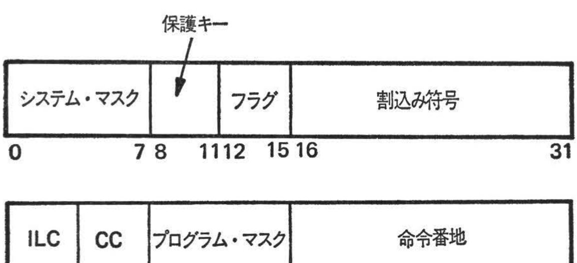
4. 外部割込み
a インタバル・タイマ終了
b オペレータ割込みボタン
c CPU-CPU 間通信割込み

5. 機械検査割込み（ハードウェアの誤動作の検査）

2.4.2 割込み機構

CPU の状態がいかにして保存され，制御が割込み処理ルーチンに移されるかを調べてみよう．CPU の状態は**プログラム状態語**（PSW）と呼ばれる 2 語長のレジスタに格納されている．PSW は，命令の実行順序を制御し，現在実行しているプログラムに関するシステムの状態を保持し，表示するために使用される．現在 CPU を制御中の PSW は“使用中の”PSW である．割込みが起こったときに PSW を格納すると，CPU の状態は調査や再ローディングのために保存することができる．新しい PSW をロードすると CPU の状態を変更することができる．

PSW の書式は次図のとおりである：



各部分については第 2.2.2.5 節で簡単に説明した．詳細に関しては IBM 370

マニュアルの“動作原理”(IBM, 2, 5) を参照のこと．

PSW のビットはある種の割込みをマスクするのに使用することができる．マスクがかけられているときには，入出力，外部，および機械検査の割込みは一時的に禁止され，保留される．15 種のプログラム割込みの内 4 種はプログラム・マスクによって無視することができる．他のプログラム割込みとスーパバイザ呼出し割込みはマスクすることができない．

プログラマは各種の状態切換え命令を使用することができる．これらには，

PSW のロード (LPSW)
プログラム・マスクの設定 (SPM)
システム・マスクの設定 (SSM)
スーパバイザ呼出し (SVC)
記憶キー設定 (SSK)

がある．これらの命令の大半は特権命令であり，スーパバイザ・モードでのみ実行できる (使用中の PSW の P フラグ=0).

5 種類の割込み——出力，プログラム，スーパバイザ呼出し，および機械検査——のおのおのに“旧”および“新”の PSW が，主記憶のあらかじめ定められた番地に格納されている．割込みが発生したとき割込み機構は，

1. 使用中の PSW を旧 PSW の番地に格納し，
2. 新 PSW の番地から PSW をロードする．

このようにして，指定された割込み処理ルーチンに制御が移され，同時に，PSW の他の多くの部分も新 PSW の内容にしたがって変更される．割込みルーチンの終了時には，

3. 通常 LPSW 命令が実行されて旧 PSW が再び PSW としてロードされる．

システムは以前の状態に戻り，割込まれたプログラムの実行が続行される．特別の割込み番地と 3 ステップの手順が 図 2.20 に示されている．

2.4.3 割込みルーチンの処理

割込みルーチンは，割込み原因を調べるために適当な旧 PSW を参照することができる．旧 PSW は割込み符号と，割込みが発生したときに実行していたプログラムの番地を納めている．

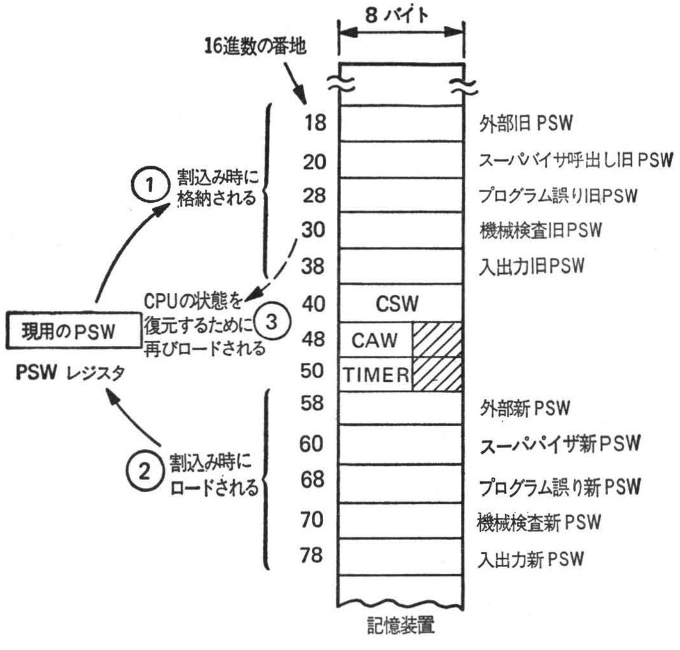


図 2.20 特別の割込み番地

プログラム割込みと外部割込みの各原因には一義的な割込み符号が定められている．これらの例は次のとおりである：

不正命令＝01
特権命令＝02
固定小数点あふれ＝08

入出力割込みが発生したとき，PSW の割込み符号は，割込みを発生したチャネルおよび装置を示す．同時に設定される CSW の状態部は割込み原因を示す情報を納めている．CSW のビット 36＝1 はチャネル動作終了（チャネル・エンド）を，ビット 37＝1 は装置動作終了（デバイス・エンド）を示す．SVC 割込みによって設定される割込み符号は，SVC 命令の直接演算子の 1 バイトに対応する．

割込みを禁止する必要のある場合が種々存在する．たとえば，カード読取り機などの装置について入出力割込みが発生したとき，現在の状態は入出力用の

旧 PSW に格納される．この割込みを処理中に他の入出力割込みが別の装置，たとえば印刷機などについて発生したとすると，状態が入出力用の旧 PSW に格納され，前の内容を破壊してしまう．その結果，保存されていた元の PSW は失われてしまい，元の状態に戻ることが不可能となる．これは次のいずれかの方法によって解決できる：

1. 1つの割込みの処理中は割込みを完全にマスクする．
2. 旧 PSW が完全に複写され別のスタックに入れられるまで，一時的に割込みをマスクする．

後者を**割込み列**に並べる処理という．その他の，割込みのマスクを必要とする場合については第 2.5 節の例を用いて説明する．

2.4.4　例外的な割込み処理の例

IBM が新しい計算機システム 371（370 より少しだけ性能がよい）を発売しようとしていると仮定する．新しい計算機は，$C(r1)^{C(r2)} \rightarrow C(r1)$ を実行する累乗命令 EXP R1, R2 を備えている以外は 370 と全く同じである．EXP の命令符号は 00 である．古い IBM 370 でこの命令符号を使用した目的プログラムを実行したい．しかし，IBM 370 が 00 なる命令符号を実行するとプログラム割込みが発生する．そこで IBM 370 を 371 のように動作させるために割込み処理プログラムを作成する．図 2.21 は処理プログラムの流れ図を示している．本節では 図 2.22 に示したように，第 2.2.5 節の EXPON ルーチンを使用することができる．

主プログラムでは，図 2.22 の文 6 で示したように，割込みに対し割込みルーチンの番地を示したプログラム割込み用の新 PSW を設定しておかなければならない．

文 30 から 79 までは割込み処理プログラムの命令である．プログラム割込みは多くの条件によって発生するので，EXP 命令が割込みを発生したことを確認するには 3 つの判定が必要である：

1. 40 番地のプログラム用旧 PSW のビット 16～31 はプログラム割込み符号 X′01′ であること．
2. プログラム用 PSW のビット 32～33 の命令長さ符号は，半語命令であることを示す符号 B′01′ であること．
3. PSW のビット 40～63 の命令番地から 2 を減じて割込まれた命令の番

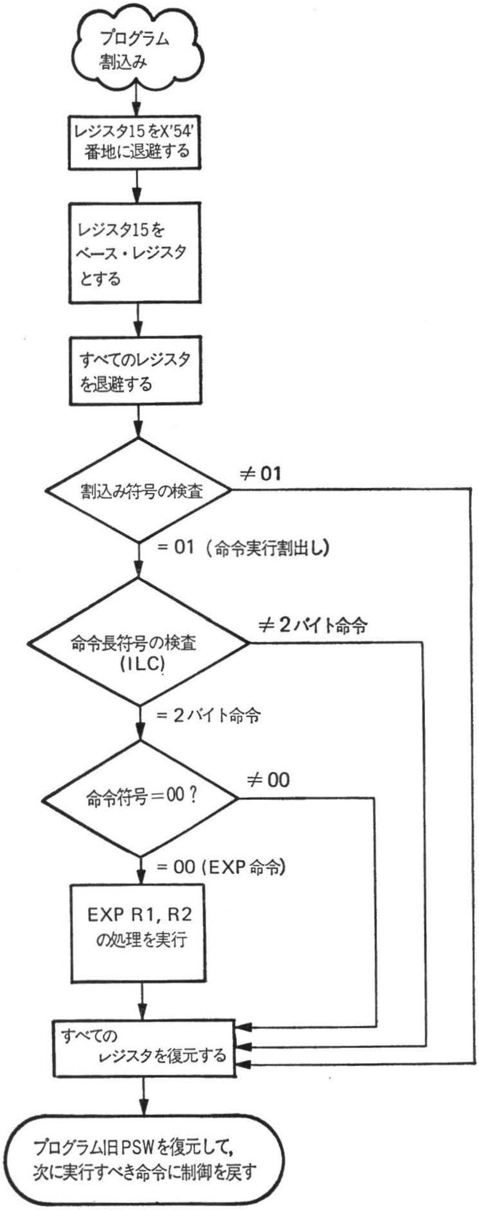


図 2.21 EXPON 割込みハンドラの流れ図

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT
001000                               1 EXPON    START X'1000'
001000 05C0                          2          BALR  12,0             SET BASE.
001002                               3          USING *,12             TELL ASSEMBLER.
000068                               4 PROGNPSW EQU   X'68'            LOCATION OF PROGRAM NEW PSW.
000028                               5 PROGOPSW EQU   X'28'            LOCATION OF PROGRAM OLD PSW.
001002 D207 0068 C106 00068 01108    6          MVC   PROGNPSW(8),=A(0,PROGINT)     SET PROGRAM NEW PSW.
                                     7 *
                                     8 *              START MAIN PROGRAM.
                                     9 *
001008 1B11                         10          SR    1,1              SET REG.1 = 0.
00100A 0011                         11          DC    X'0011'          PERFORM 'EXP 1,1' (0**0=1).
00100C 5010 C02A          0102C     12          ST    1,RESULT1        SAVE RESULT.
001010 4120 0002          00002     13          LA    2,2              SET REG.2 = 2.
001014 4130 0004          00004     14          LA    3,4              SET REG.3 = 4.
001018 0023                         15          DC    X'0023'          PERFORM 'EXP 2,3' (2**4=16).
00101A 5020 C02E          01030     16          ST    2,RESULT2        SAVE RESULT.
00101E 5830 C10E          C1110     17          L     3,=F'-2'         SET REG.3 = -2.
001022 0023                         18          DC    X'0023'          PERFORM 'EXP 2,3' (16**-2=0).
001024 5020 C032          01034     19          ST    2,RESULT3        SAVE RESULT.
001028 07FE                         20          BR    14               RETURN TO CALLER.
                                    21 *
                                    22 *        SAVE RESULTS HERE.
                                    23 *
00102C                              24 RESULT1  DS    F                ** RESULT SHOULD BE 1.
001030                              25 RESULT2  DS    F                ** RESULT SHOULD BE 16.
001034                              26 RESULT3  DS    F                ** RESULT SHOULD BE 0.
                                    27 *
                                    28 *        EXP PROGRAM INTERRUPT HANDLER.
                                    29 *
001038 5CF0 C054          00054     30 PROGINT  ST    15,X'54'         SAVE REG.15 IN UNUSED LOCATION.
00103C 05F0                         31          BALR  15,0             SET UP BASE REGISTER.
001C3E                              32          USING *,15             ...
00103E 900E F086          010C4     33          STM   0,14,SAVEAREA    SAVE REGISTER 0-14.
001042 D203 FCC2 0054 01100 00054   34          MVC   SAVEAREA+15*4(4),X'54'    MOVE REG.15 INTO SAVEAREA.
                                    35 *              ** TEST IF EXP PROGRAM INTERRUPT.
001048 9501 002B    0002B           36          CLI   PRCGOPSW+3,X'01'    IS IT OPERATION EXCEPTION?
00104C 4770 F07C          010BA     37          BNE   RETURN           NO, IGNORE INTERRUPT.
```

図 2.22 EXPON プログラム割込みハンドラ

```
001050 1B22                             38          SR     2,2             GET INSTRUCTION LENGTH FIELD
001052 4320 002C               0002C    39          IC     2,PROGOPSW+4      FROM OLD PSW.
001056 8820 0006               00006    40          SRL    2,6                   ...
00105A 8920 0001               00001    41          SLL    2,1                   ...
00105E 5920 F0D6               01114    42          C      2,=F'2'         WAS IT A 2-BYTE INSTRUCTION?
001062 4770 FC7C               010BA    43          BNE    RETURN          NO, IGNORE INTERRUPT.
001066 5810 002C               0002C    44          L      1,PROGOPSW+4    GET ADDRESS OF NEXT INSTRUCTION.
00106A 1B12                             45          SR     1,2             MOVE BACK 2 BYTES TO INTERRUPTED INST.
00106C 9500 1000         00000          46          CLI    0(1),X'00'      IS IT EXP INSTRUCTION ('00' OP)?
001070 4770 F07C               010BA    47          BNE    RETURN          NO, IGNORE INTERRUPT.
                                        48 *               ** PERFORM EXP SIMULATION NOW.
001074 1B22                             49          SR     2,2             ZERO OUT REGISTER 2.
001076 4320 1001               00001    50          IC     2,1(0,1)        LOAD BYTE 2 OF INSTRUCTION.
C0107A 8C20 0004               00004    51          SRDL   2,4             SHIFT EXP REGISTER INTO REG 3.
00107E 8830 001C               0001C    52          SRL    3,28            RIGHT SHIFT EXP REG IN REQ 3.
000002                                  53 RBASE    EQU    2               REG 2 CONTAINS BASE REG NO.
000003                                  54 REXP     EQU    3               REG 3 CONTAINS THE EXP REG NO.
000004                                  55 BASE     EQU    4               REG 4 CONTAINS THE BASE.
000005                                  56 EXP      EQU    5               REG 5 CONTAINS THE EXPONENT.
001C82 8920 0002               00002    57          SLL    RBASE,2         MULTIPLY BASE REG NO. BY 4.
001086 8930 0002               C0002    58          SLL    REXP,2          MULTIPLY EXP REG NO. BY 4.
00108A 5842 F086               010C4    59          L      BASE,SAVEAREA(RBASE)     FETCH THE BASE.
00108E 5853 F086               010C4    60          L      EXP,SAVEAREA(REXP)  FETCH THE EXPONENT.
001092 1255                             61          LTR    EXP,EXP         TEST IF EXP NEG, ZERO, POS.
001094 4740 F06E               010AC    62          BM     NEG             TRANSFER IF NEGATIVE EXPONENT.
001098 4780 F074               010B2    63          BZ     ZERO            TRANSFER IF ZERO EXPONENT.
00109C 1874                             64          LR     7,BASE          INITIALIZE RESULT.
00109E 47F0 F066               010A4    65          B      ENDLOOP         MULTIPLY EXP TIMES.
0010A2 1C64                             66 LOOP     MR     6,BASE          MULTIPLY.
0010A4 4650 F064               010A2    67 ENDLOOP  BCT    EXP,LOOP        LOOP EXP NO. OF TIMES.
0010A8 47F0 F078               010B6    68          B      DONE
0010AC 1B77                             69 NEG      SR     7,7             IF NEG EXP, RESULT = 0.
0010AE 47F0 F078               010B6    70          B      DONE
0010B2 4170 0001               00001    71 ZERO     LA     7,1             IF ZERO EXP, RESULT = 1.
0010B6 5072 F086               C10C4    72 DONE     ST     7,SAVEAREA(RBASE)   PUT ANSWER IN SAVE AREA REG.
0010BA 980F F086               010C4    73 RETURN   LM     0,15,SAVEAREA   RESTORE THE REGISTERS.
0010BE 8200 0028         00028          74          LPSW   PROGOPSW        RETURN TO INSTRUCTION AFTER INTERRUPT.
0010C4                                  75 SAVEAREA DS     16F             SPACE TO SAVE THE 16 REGISTERS.
                                        76          END
001108 0000000000001C38                 77                 =A(0,PROGINT)
001110 FFFFFFFE                         78                 =F'-2'
001114 00000002                         79                 =F'2'
```

図 2.22 EXPON プログラム割込みハンドラ（続き）

地を求め，命令符号を検査すること．

なお，X '54' 番地はベース・レジスタを設定するために使用されている．プログラム用の旧 PSW を文 74 でロードし直すと制御は割込まれたプログラムに返却される．

2.4.5　非同期割込み処理の例

第 2.3.6 および第 2.3.7 節のカード読込み／印刷のプログラムを再び考えてみよう．これらのプログラムでは，どちらも CPU を 100% 使用していた．しかし，一体何をしていたのであろうか．CPU は入出力が完了するのを待合せて，入出力状態を判定し続けていたのである (TIO)．したがって実際には 99% 以上の時間が空費されていた．本節の例では，割込み機能を利用して CPU 時間のより有効な利用をはかりたい．こうすると CPU は大半の時間を計算処理のために使用することができる．“次の入出力を開始すべきである”と割込みが通知してきたときには，CPU は即座に入出力を起動し，再び計算に移る．図 2.23 はこの方法を示している．

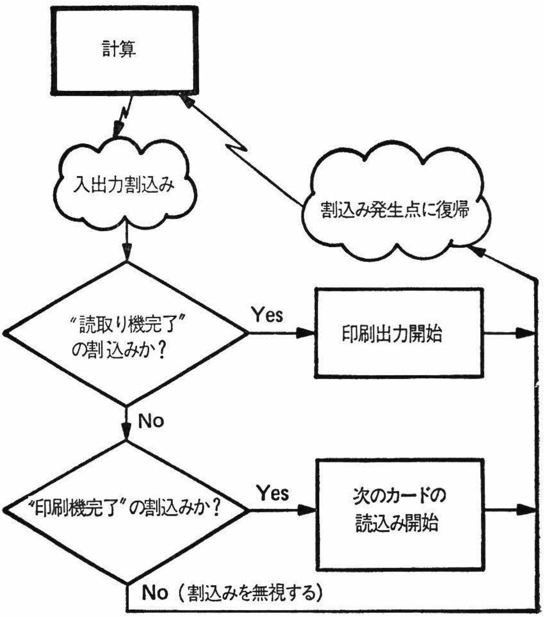


図 2.23　割込み駆動型入出力処理

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

001000                                1 COUNT    START X'1000'
001000 05F0                           2          BALR  15,0             SET BASE.
001002                                3          USING *,15             TELL ASSEMBLER.
000048                                4 CAW      EQU   X'48'            LOCATION OF CHANNEL ADDRESS WORD.
000040                                5 CSW      EQU   X'40'            LOCATION OF CHANNEL STATUS WORD.
000078                                6 IONPSW   EQU   X'78'            LOCATION OF I/O NEW PSW.
000038                                7 IOOPSW   EQU   X'38'            LOCATION OF I/O OLD PSW.
001002 D207 0078 F0CE 00078 010D0     8          MVC   IONPSW(8),=A(0,INT) SET I/O NEW PSW (INTERRUPT ADDR).
001008 D203 0048 F0D6 00048 010D8     9          MVC   CAW(4),=A(READ1)    SET CAW TO READ A CARD.
00100E 8000 F0EA      010EC          10          SSM   =X'FF'           ENABLE INTERRUPTS.
001012 9C00 000C      0000C          11          SIO   X'00C'           START READER FOR FIRST CARD.
                                     12 *
                                     13 *              START COMPUTATION.
                                     14 *
001016 5810 F0DA            010DC    15          L     1,=F'1000000'    START COUNTING FROM 1 TO
00101A 1B00                          16          SR    0,0              1000000.
00101C 5A00 F0DE            010E0    17 LOOP     A     0,=F'1'          (ADD 1).
001020 4610 F01A            0101C    18          BCT   1,LOOP           (DECREMENT COUNTER UNTIL 1000000).
001024 07FE                          19          BR    14               FINALLY FINISHED, RETURN TO CALLER.

                                     20 *
                                     21 *              INTERRUPT HANDLER.
                                     22 *
001026 50F0 0054            00054    23 INT      ST    15,X'54'         SAVE REG 15 AT UNUSED LOCATION.
00102A 05F0                          24          BALR  15,0             SET UP INTERRUPT HANDLER BASE REG.
00102C                               25          USING *,15             ...
00102C 9104 0044      00044          26          TM    CSW+4,X'04'      WAS THIS A DEVICE END INTERRUPT?
001030 4780 F01C            01048    27          BZ    RETURN               NO, IGNORE THIS INTERRUPT.
001034 D501 003A F0BC 0003A 010E8    28          CLC   IOOPSW+2(2),=X'000C'     IS IT READER INTERRUPT?
00103A 4780 F024            01050    29          BE    READINT              YES, READER INTERRUPT.
00103E D501 003A F0BE 0003A 010EA    30          CLC   IOOPSW+2(2),=X'000E'     IS IT PRINTER INTERRUPT?
001044 4780 F032            0105E    31          BE    PRINTINT             YES, PRINTER INTERRUPT.
001048 58F0 0054            00054    32 RETURN   L     15,X'54'         RESTORE ORIGINAL CONTENTS OF REG 15
00104C 8200 0038      00038          33          LPSW  IOOPSW              AND I/O OLD PSW (RETURN TO MAIN PROG).
                                     34 *              ** READER INTERRUPT HANDLER.
001050 D203 0048 F0B8 00048 010E4    35 READINT  MVC   CAW(4),=A(PRINT1)    START PRINTER AND PRINT CARD
001056 9C00 000E      0000E          36          SIO   X'00E'                   JUST READ.
```

図 2.24　1,000,000 まで数を数えながら単一バッファリングで入出力を行なうプログラム

```
00105A 47F0 F01C             01048    37            B      RETURN          RETURN TO POINT OF INTERRUPTICN.
                                      38 *                 ** PRINTER INTERRUPT HANDLER.
00105E D203 0048 F0AC 00048 01CD8    39 PRINTINT   MVC    CAW(4),=A(READ1)     START READER TO GET
001064 9C00 000C         0000C        40            SIO    X'00C'                     NEXT CARD.
001068 47F0 F01C             01048    41            B      RETURN          RETURN TO POINT OF INTERRUPTION.
                                      42 *
                                      43 *
00106C 0C000000
001070 0200108000000050               44 READ1      CCW    X'02',BUFFER1,X'00',80   READ 80-BYTE CARD.
001078 0900108000000050               45 PRINT1     CCW    X'09',BUFFER1,X'00',80   PRINT 80-BYTE CARD.
C01080                                46 BUFFER1    DS     CL80            BUFFER FOR 80-BYTE CARD.
                                      47            END
0010D0 0C0000C000001026               48                   =A(0,INT)
0010D8 00001070                       49                   =A(READ1)
0010DC 000F4240                       50                   =F'1000000'
C010E0 0C000001                       51                   =F'1'
0010E4 00001078                       52                   =A(PRINT1)
0010E8 000C                           53                   =X'000C'
0010EA 000E                           54                   =X'000E'
0010EC FF                             55                   =X'FF'
```

図 2.24　1,000,000 まで数を数えながら単一バッファリングで入出力を行なうプログラム（続き）

この方法を用いたプログラムは 図 2.24 に示されている．この例では，“有用な計算”として1から1,000,000までを数えている．一般には，利用者はもっと内容のある計算を行なおうとしている．前の例と同様に，プログラマは割込み処理ルーチンの初期化を行なわねばならない．これを次に示す：

1. 入出力割込み処理ルーチンの番地が設定された入出力割込み用の新 PSW を設定する．
2. カードを読込む最初の入出力を起動し，ポンプに“呼び水”を行なう．
3. 入出力割込みを受理するためにシステム・マスクを設定する．

これらの手順が完了すると，計算命令 15～19 を開始することができ，入出力は完全に割込みハンドラによって行なわれる．この方法はしばしば**割込み駆動型処理**と呼ばれる．

割込みハンドラは文 20～55 から構成されている．一般には，割込みハンドラは物理的に別個の利用者サブルーチンまたはオペレーティング・システムの一部である．したがって，割込みハンドラでは，前の例で示したように，正しい番地に自らのベース・レジスタを設定する必要がある．

入出力割込みの原因は多数存在する，誤りの発生でなく．正常動作の場合にも複数の割込みが発生する．これらには次のものがある：

1. チャネル動作終了割込み
2. 制御装置動作終了割込み
3. 装置動作終了割込み

チャネルは装置の動作が完了する前に，その動作を完了することが可能である．たとえば印刷機を考えてみよう．チャネルは，主記憶から印刷機の内部バッファにデータ・バイトを転送しなければならない．もしこれが入出力プログラムの最後のチャネル・コマンド語である場合には，チャネルをこれ以上使用する必要がないので，チャネル動作終了割込みが発生する．印刷機は，単独で，バッファの内部によって最後の行を印刷し，適当に紙送りを行なうことができる．印刷機でこれらの仕事が終了したとき，装置動作終了割込みが発生する．これらの割込みの種類や発生の順序は，使用する装置，制御装置およびチャネルの型によって定まっている――これらについては個別に述べたマニュアルが用意されている．本節の印刷機の例では，印刷機がすでに動作中でないことを通知する装置動作終了割込みを待合せる必要がある．この情報は，文 26 および 27 で示されているように，CSW から求められる．

割込みの種類（デバイス・エンド）を確認すると次に割込みを発生した装置を決定しなければならない．この情報は入出力用の旧 PSW の割込み符号に入っている．装置終了割込みが読取り機（X′00C′）で起これば，印刷機で入出力を開始する．印刷機（X′00E′）で起これば，読取り機で入出力を開始する．他の装置による割込みは無視される．

この例では，割込みハンドラは各割込みを処理するのに約 13 命令を実行するだけである．処理装置の速度が毎秒 100 万命令とすると，割込み処理時間は 13 マイクロ秒である．読取り機と印刷機は，毎分 1,000 枚の速度として，各動作を 60 ミリ秒かかって終了する．したがって，入出力処理は 60 ミリ秒ごとに 13 マイクロ秒の CPU 時間を使用する．すなわち，入出力処理は CPU 時間を 0.02% だけ使用し，99.98% は他の計算のために使用することができる．

ダブル・バッファリングを行なうともっと効率のよい入出力ハンドラを作成することができ，カード読込み/カード印刷の速度を毎分 500 行から毎分 1,000 行に上げることができる．また，複数のチャネル・コマンド語で構成される入出力プログラムを用いると，割込みの回数を 60 ミリ秒に 1 回から 3.6 秒に 1 回の割合に減少させることができる．これらの技法は第 2.3.7 節で概説した．

実際には，入出力ハンドラは，誤り条件や入出力の停止などのほかの事柄についても処理する必要がある．より完全な例を次節にて示す．

2.5　入出力および割込み処理プログラムの例

本章を締めくくるために，2つの有益な例を簡単に論じよう．

2.5.1　IPL プログラムの例

前節の例においては，プログラムはすでに記憶装置に入っていると仮定していた．プログラムはどのようにして記憶装置に入れられたのであろうか．処理装置の制御パネルのトグル・スイッチやノブを使用して入力することもできるが，あまり手際のよい方法ではないし速度も遅い．ローダを使用したらよいと考える人もあるだろう．しかし，ローダをどのようにして入力するのか．オペレーティング・システムは使用できないだろうか．しかし，オペレーティング・システムはどのようにして入力するのか．全く方法はないのだろうか．その答えが初期プログラム・ロード（IPL）・プログラムである（IBM 3, 4)．実際の IPL プログラムの流れ図は 図 2.25 に，対応するコードは 図 2.26 に示されている．このプログラムは 80 バイトのカード 4 枚に納まる．このプログラ

ムをどのようにして記憶装置に入れたらよいだろうか．幸いなことに計算機のハードウェアに多少の用意がしてある．計算機のオペレータが IPL ボタンを押すと次の動作が自動的に遂行される：

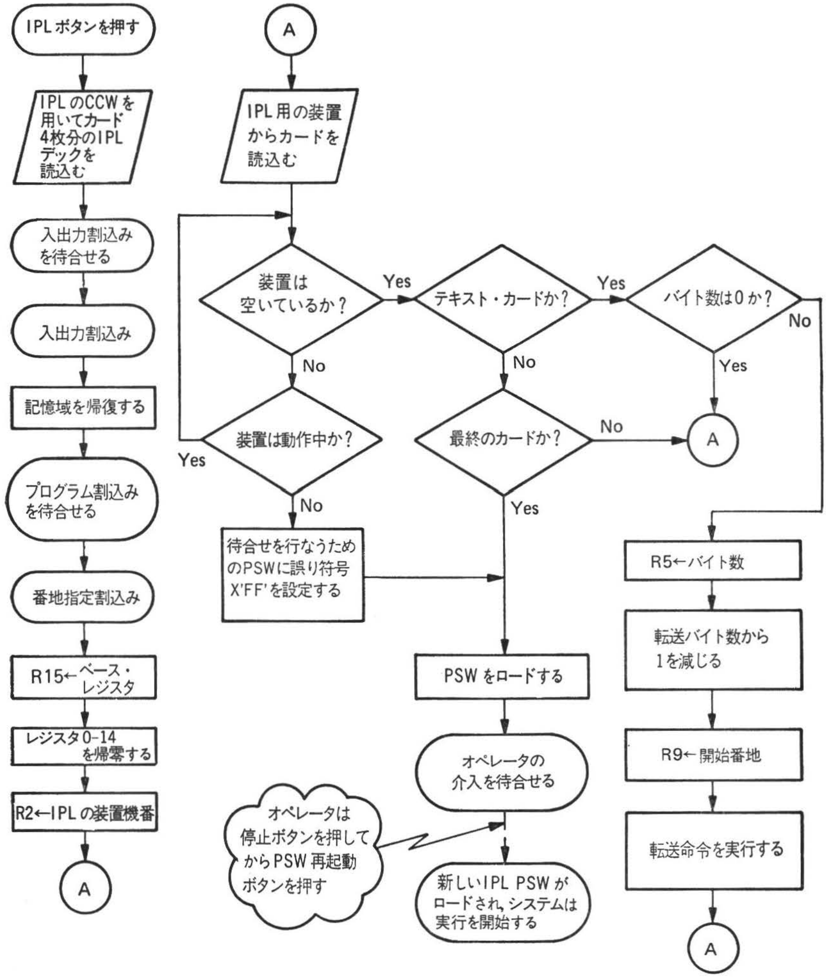


図 2.25 例題オペレーティング・システムで使用されている IPL プログラムの流れ図

1. IPL装置機番スイッチで選択された装置に入っている第1レコードから24バイトが読込まれる（すなわち，計算機はカード読取り機の最初のカードから24バイト読込む）．これらのバイトは0〜23番地に格納される．
2. IPL装置機番および8という値のCAW（すなわち最初のCCWとして8番地が使用される）を用いてSIO命令が実行される．入出力プログラムは複数のCCWを使用してもよい．
3. 入出力プログラムが完了すると，0〜8番地の8バイトが入出力用旧/新PSWであるものとして使用される．すなわち，割込み符号は2〜3番地に格納されて，この8バイトがPSWとして使用されることとなる．

ここに述べた手順はIBM 360および370に関するものであるが，たいていの計算機でも同様の方法が用いられている．

図2.26のIPLプログラムでは，初期化の入出力プログラムは5つのCCWから成っている．注目すべきことは，入出力が開始されたときには，最後のCCWは主記憶に入っていない．しかし，これが必要となるまでに最初のCCWでロードされるならばプログラムは正しく働く．この過程はしばしば**ブートストラップ・ローディング**と呼ばれる（小さな部分でより大きな部分をロードし，それでまたより大きな部分をロードする……）．

このIPLプログラムは割込み機構を利用して記憶装置の内容をすべて抹消している．すなわち，プログラムは番地侵害を起こすまで0を格納する．

このIPLプログラムは標準のIBMアセンブラによって生成された目的プログラムをロードするように作られている．目的プログラムはオペレーティング・システム自体のこともあるし，前の例のように，オペレーティング・システムが不要のときは利用者プログラムのこともある．

利用者のプログラムがロードされると，IPLプログラムは停止して外部割込みが起こるのを待合せる．計算機のオペレータはこの機会にコンソールの**トグル・スイッチ**を用いてプログラムなどの変更を行なうことができる．用意が整ったときには，オペレータはSTOPボタン，次にPSW再起動ボタンを押す．IPLプログラムは利用者プログラムに制御を移す．

同一の過程はオペレーティング・システムをロードするのにも使用できる．この場合には通常，カード読取り機ではなくてディスクやテープ駆動装置がIPL装置として使用される．しかし全体のアルゴリズムは同じである．

```
  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT

000000                                 1 IPLPROG  CSECT
000000                                 2          USING *,0             ESTABLISH ADDRESSIBILITY
                                       3 *        CARD 1 CONTAINS ONLY THE NEXT 24 BYTES
000000 0000000000000070                4 IPLPSW   DC    X'00000000',A(START)
000008 0200001840000050                5 CCW1     CCW   X'02',CCW3,X'40',80
000010 0200006040000050                6 CCW2     CCW   X'02',NEWPROG-8,X'40',80
                                       7 *        CARD 2 STARTS HERE
000018 020000A800000050                8 CCW3     CCW   X'02',NEWPROG+64,X'00',80
000020 0002000000000000                9 DEAD     DC    X'0002000000000000'  FINAL WAIT PSW
000040                                10 CSW      EQU   64              CHANNEL STATUS WORD (CSW).
000048                                11          ORG   *+X'20'
000048 000000E0                       12 CAW      DC    A(READCCW)      CHANNEL ADDRESS WORD (CAW).
00004C                                13          DS    F
000050 FFFFFFFF                       14 TIMER    DC    F'-1'           TIMER (INITIALLY SET TO -1).
000068                                15          ORG   *+X'14'
                                      16 *        CARD 3 STARTS HERE
000068 0000000000000088               17 NEWPROG  DC    F'0',A(L2)      FOR ADDRESS INTERRUPT CAUSED
000070 05F0                           18 START    BALR  15,0               BY ZEROING CORE.
000072                                19          USING *,15            BASE REGISTER 15
000072 4110 F076            000E8     20          LA    1,BUFFER        GET ADDRESS OF END OF
000076 9200 F076      000E8           21          MVI   BUFFER,X'00'       THIS PROGRAM.
00007A D2FF 1001 1000 00001 00000     22 L1       MVC   1(256,1),0(1)   ZERO CORE UNTIL YOU CAUSE
000080 4111 0100            00100     23          LA    1,256(1)           AN ADDRESSING INTERRUPT.
000084 47F0 F008            0007A     24          B     L1
000088 980E 0400            00400     25 L2       LM    0,14,X'400'     CLEAR REGISTERS
00008C 4820 0002            00002     26          LH    2,2             GET IPL UNIT ADDRESS IN REG 2
000090 9C00 2000      00000           27 LL1      SIO   0(2)            READ IN THE OBJECT DECK
000094 9D00 2000      00000           28 L3       TIO   0(2)            IF AVAILABLE, SEE WHAT KIND
000098 4780 F03A            000AC     29          BE    L4                 OF CARD WAS READ.
00009C 9102 0044      00044           30          TM    CSW+4,X'02'     IF EXCEPTION, SET FLAG
0000A0 47E0 F022            00094     31          BNO   L3                 AND STOP.
0000A4 92FF 0027      00027           32          MVI   DEAD+7,X'FF'    SET ERROR CODE IN PSW
```

図 2.26 IPL プログラムの例

```
                                   33 *          CARD 4 STARTS HERE
0000A8 8200 0020      00020        34 L6         LPSW  DEAD            LOAD PSW TO WAIT FOR OPERATOR.
0000AC 95E7 F078      000EA        35 L4         CLI   BUFFER+2,C'X'   IS IT A TXT CARD?
0000B0 4780 FC4E            000C0  36            BE    L5
0000B4 95D5 F078      000EA        37            CLI   BUFFER+2,C'N'   IS IT AN END CARD?
0000B8 4780 F036            000A8  38            BE    L6
0000BC 47F0 F01E            00090  39            B     LL1             IGNORE ALL OTHERS.
0000C0 9500 F081      000F3        40 L5         CLI   BUFFER+11,X'00' IF ZERO COUNT, IGNORE
0000C4 4780 F01E            00090  41            BE    LL1                THIS CARD.
0000C8 4350 F081            000F3  42            IC    5,BUFFER+11     OTHERWISE, GET COUNT FOR
0000CC 0650                        43            BCTR  5,0                MOVING TEXT INTO CORE AT THE
0000CE 5890 F07A            000EC  44            L     9,BUFFER+4         CORRECT LOCATION.
0000D2 4450 F068            000DA  45            EX    5,MOVE          MOVE TEXT.
0000D6 47F0 F01E            00090  46            B     LL1             GET ANOTHER CARD.
0000DA D200 9000 F086 00000 000F8  47 MOVE       MVC   0(0,9),BUFFER+16
0000E0 020000E800000050            48 READCCW    CCW   X'02',BUFFER,X'00',80
0000E8                             49 BUFFER     DS    CL80            CARD BUFFER.
                                   50            END
```

図 2.26　IPL プログラムの例（続き）

2.5.2 入出力バッファリングの例

前述の割込み処理の例はむしろ単純なものであった．例外条件による割込みはプログラムに同期して発生し，従来のサブルーチン呼出しと同様に処理された．また，非同期の入出力の例は計算プログラムと完全に独立に実行された．計算プログラムと割込み処理プログラムとが通信を行なわなければならない場合には，幾つかの興味ある問題が存在する．次に示す例がその問題点を示している．さらに，それらの問題は後の章で示されるプロセス通信と同期および装置のスケジューリングの基礎となる．

2.5.2.1 印刷機用のユーティリティ・ルーチン

第1章で述べたように，利用者プログラムで SIO や TIO などの入出力命令を直接実行することは滅多にない．入出力処理は，オペレーティング・システムで準備され，利用者プログラムにとってこれらは特別のサブルーチンや SVC のように見える．本節の例では単純な印刷機用のユーティリティ・ルーチンを示す．1行の印刷はわずか2，3命令で行なうことができるが，前の例で見たように，処理装置の利用効率や入出力の仕事量を最大とするために，ずっと複雑なルーチンが必要となる（たとえば，ダブル・バッファリングや割込み駆動型処理）．第5章ではこの問題を詳細に述べる．

本節の例の印刷用パッケージでは，2つの入口を持つ簡単なサブルーチンが使用される．入口 PRINTSET は印刷用パッケージを初期化するために1回だけ呼出される．それ以後1行の印刷を行ないたいときには入口 PRINT が呼出される．

2.5.2.2 例題プログラムの構造

この例は5つの副プログラムから構成されている．プログラム MAIN は印刷用パッケージを使用する典型的な利用者プログラムを示している．ここでは MAIN は乱数を発生し印刷する（図 2.28 の出力例を参照）．

印刷用のパッケージは次の4つのサブプログラムから構成されている：

1. PRINTSET　このサブプログラムは MAIN から1回だけ呼出され，変数を初期化し，入出力用の新 PSW を設定する．
2. PRINT　このサブプログラムは各行を印刷するために MAIN から呼出される．印刷機が“準備完了”のときは直ちに入出力が起動され，

以前の要求を処理中で“動作中”のときは，後刻“準備完了”となったとき入出力が開始されるように，新しい要求を行列に加えておく．いずれの場合にもこれらの処理を終えて呼出し者に復帰する．

3. PRINTINT　このサブプログラムは入出力割込みの処理を行なう．また，誤りが発生していないか検査する．誤りが発生していれば，再度入出力要求を出して回復を試みる．誤りがなく，PRINT によって出力要求が行列に加えられていれば入出力が起動される．

4. SIOSTART　これは PRINT および PRINTINT の両者によって使用される共通ルーチンである．SIOSTART は適当な CCW を設定してから，入出力を起動するために SIO 命令を実行する．

この例題の印刷用パッケージの構造は図 2.27 に示されている．共通のデータ・セットは，要求の状況を把握したりルーチン間の通信や同期を行なうために，印刷用パッケージの4つのルーチン全部によって共用される．図 2.27 にはこれらのデータ要素も示されている．

前述したように，この印刷用パッケージはダブル・バッファリングの技法を利用している．したがって，2つのバッファ BUFFER1 と BUFFER2，および各バッファに関する次の4つの情報が使用されている：

1. BUFFNXT　他のバッファへの番地ポインタ．
2. BUFFUSE　フラグ（0または1）．
   1＝バッファは印刷待ちまたは印刷中の情報を格納している（バッファが“一杯”である）．
   0＝バッファの内容は印刷され，バッファを再使用することができる（バッファが“空”である）．
3. BUFFLNG　バッファに内容が入っているとき，この変数は印刷すべき行の長さを示す（1～132 バイト）．
4. BUFF　バッファに内容が入っているとき，この変数が印刷すべき実際の内容の写しである（最大 132 バイトの長さ）．

このバッファ情報のほかに次の4つの変数が用いられている：

1. IOSWITCH　印刷機で入出力が実行中のとき，IOSWITCH は印刷中のバッファ――BUFFER1 または BUFFER2――へのポインタである（したがって，IOSWITCH が 0 でなければ入出力は“活動中”である）．印刷機で入出力が実行中でないときには IOSWITCH は 0 で

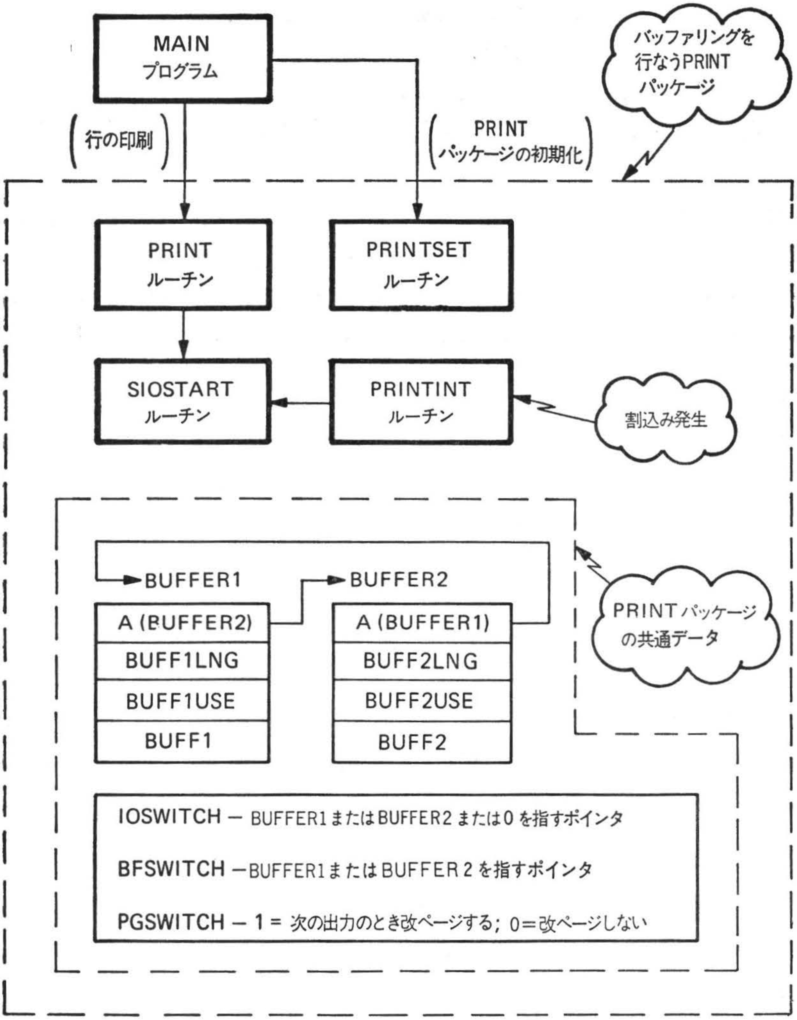


図 2.27 印刷用パッケージの構造

ある（この場合入出力は活動中でないという）.

2. BFSWITCH　BFSWITCH は BUFFER1 と BUFFER2 とを交互に切換えるポインタである．BFSWITCH は，利用者の次の印刷要求がどちらのバッファを用いてなされるべきかを指定する．

3. PGSWITCH　フラグである（0または1）.
1＝改ページを行なう.
0＝通常の1行送りを行なう.
4. CCSWITCH　現在実行中のCCWがあれば，その先頭を指すポインタである．これは入出力誤り処理と再実行のときにのみ使用される．

このプログラムは，オペレーティング・システム全体と比べるとはるかに単純であるが，多数の基本的な問題点を含んでいるので注意深く学んでほしい．なお，このプログラムは注意して作成し，デバッグしてあるが，どんな場合でも適用可能であることの保証や効率よく働く保証はない．

2.5.2.3 プログラム MAIN

プログラム MAIN は，印刷パッケージの試験のための“擬似”プログラムである．この例では，200個の乱数を発生して印刷する．各乱数は0と100の間の数である．基本のアルゴリズムは次式である：

$$r_{i+1}=23\times r_i+17(\mathrm{mod}\ 2^{32}-1)$$

実際の乱数 R は r を 0～100の範囲に正規化して計算される．乱数の質をよくするために，前の2つの乱数 R_{j-1} と R_{j-2} の積の個数だけ r_j をスキップして R_j を求める．MAIN の流れ図と出力例は 図 2.28 に示されている．前述したように，ここで問題としているのは印刷パッケージであって，プログラム MAIN の性質とは関係がない．

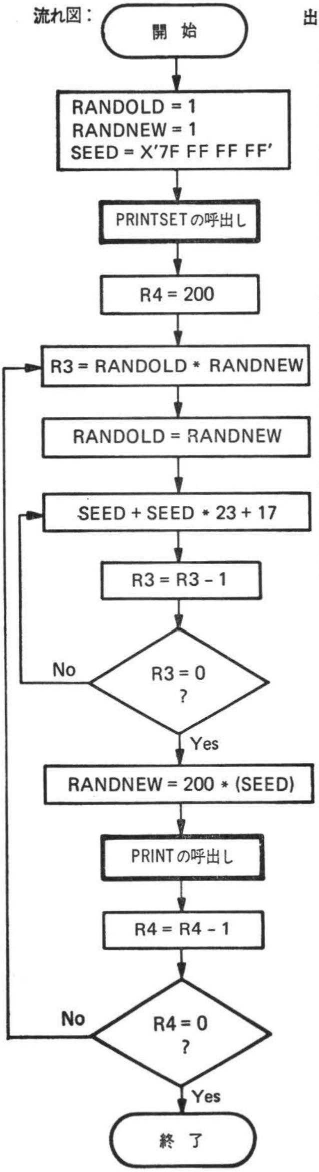
2.5.2.4 PRINTSET ルーチン

PRINTSET は 図 2.29 に示したように単純で明解である．このルーチンはスイッチを初期化して，両方のバッファが空で入出力が活動中でない旨の表示を行なう．入出力用の新 PSW(78_{16}番地）を設定して，入出力割込みが発生したとき制御がPRINTINT に移されるようにする．すべての条件が初期化されたら最後に，割込みを可能とする．

2.5.2.5 PRINT ルーチン

1行の印刷を行なうために呼出されると，PRINT は次に使用すべきバッファ（ポインタ BFSWITCH で指定されている）を検査する．バッファがすでに一杯であれば（すなわち，前の要求によって入出力行列に加えられていると），

流れ図：



出力：

```
00001 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 099.
00099 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 015.
01485 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 051.
00765 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 012.
00612 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 022.
00264 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 093.
02046 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 018.
01674 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 052.
00936 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 077.
04004 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 010.
00770 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 030.
00300 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 011.
00330 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 052.
00572 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 047.
02444 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 059.
02773 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 062.
03658 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 080.
04960 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 054.
04320 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 015.
00810 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 039.
00585 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 009.
00351 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 009.
00081 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 020.
00180 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 094.
01880 RANDOM NUMBERS SKIPPED. NEXT NUMBER IS 023.
```

(次の乱数SEEDを計算する)

(RANDNEW は0と100の間の乱数)

図 2.28 MAIN プログラム（乱数を発生し印刷する）

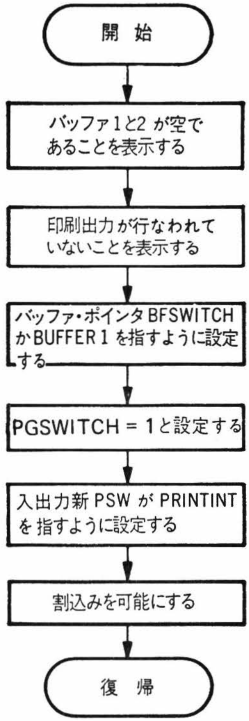


図 2.29　PRINTSET ルーチン（初期化）

PRINT は LPSW を設定して入出力が完了するまで“待合せ”なければならない．バッファが空であるか，または入出力完了で空になったときには，印刷すべき行と長さが選択されたバッファ領域に移される．そしてこのバッファは一杯であるとの印がつけられる．もう一方のバッファの内容を印刷するために入出力が実行中であれば，PRINT は直ちに呼出者に復帰する．これに対して，印刷機が何の動作もしていないときには，PRINT は SIOSTART ルーチンを呼出して必要な CCW を設定し，入出力を起動させる．そして呼出者に復帰する．図 2.30 は PRINT の流れ図を示す．

2.5.2.6　SIOSTART ルーチン

SIOSTART ルーチンは，図 2.31 図に示したように，入出力を起動したり

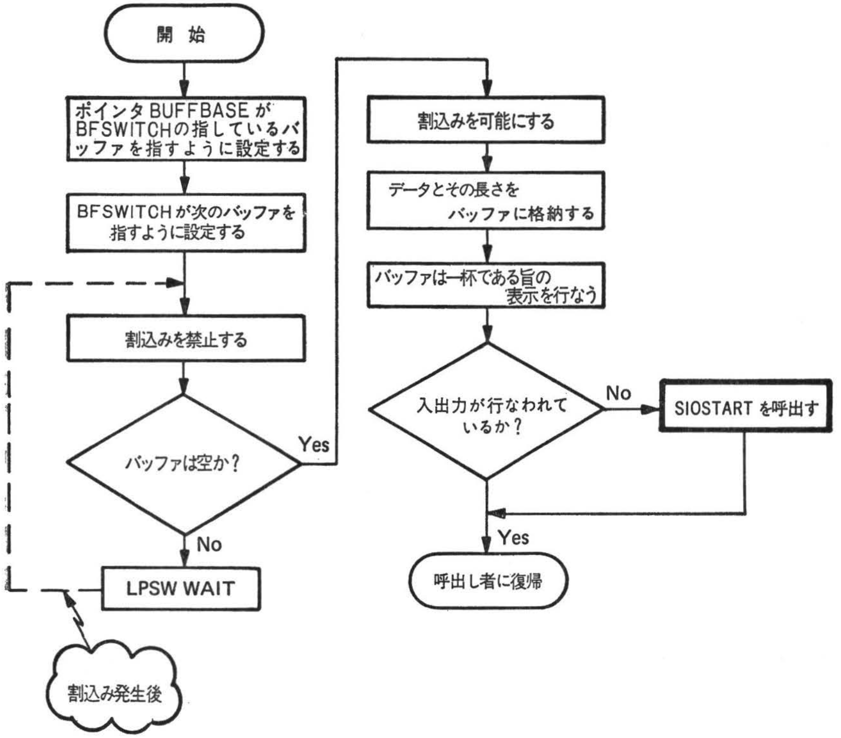


図 2.30 印刷ルーチン

誤り状態の後で入出力を再起動したりするユーティリティ・ルーチンである．このルーチンは，入出力が活動中でバッファを使用中であることを示すためにIOSWITCH ポインタを設定する．印刷機が動作中であれば SIOSTART が呼出されることはないので，印刷機が準備完了しているかどうかの検査は単に用心のためである．SIOSTART は印刷用の CCW (PRINTCCW) を修飾して印刷すべきバッファと長さを指定する．132 バイト以外の長さ指定も行なわれるので，長さ検査抑制ビットが使用される．改ページが要求されているときには (PGSWITCH=1)，CAW は，PRINTCCW の前に置かれた改ページを行なう CCW (EJECTCCW) を指すように設定され，PGSWITCH は 0 にリセットされる．それ以外のとき，CAW は PRINTCCW を指すように設

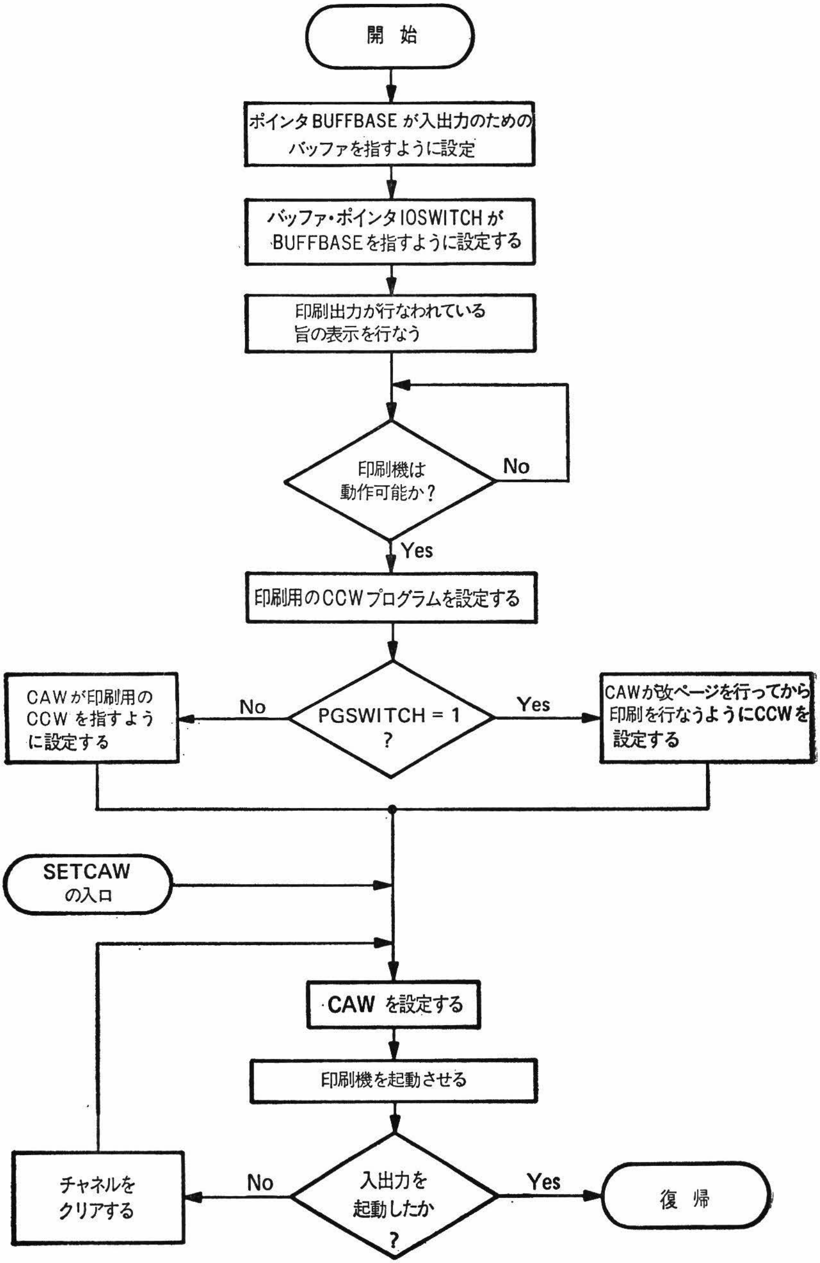


図 2.31　SIOSTART ユーティリティ・ルーチン

定される．

これらの手順がすべて完了した後に SIO 命令が実行される．これがチャネルおよび装置に受理されると SIOSTART は呼出し者に復帰する．受理されなければ，TIO ループを回って装置が準備完了となるのを待合せ，SIO を再び試みる．

2.5.2.7 PRINTINT ルーチン

入出力割込みが発生すると PRINTINT に制御が移され，すべてのレジスタを退避してからベース・レジスタを設定する．この割込みが印刷機以外からのものであったり，現在入出力中の印刷機が存在しないときには，この割込みは無視される．CSW の状態ビット（装置検査，データ検査など）から誤り発生が判明したときには入口 SETCAW を経由して SIOSTART ルーチンに行き，入出力を再起動して，次に割込みが発生するまで制御を割込まれたルーチンに返却する．CSW に“ページ終了”表示子が設定されていれば，PGSWITCH を 1 に設定して次の印刷のときに改ページされるようにする．

入出力割込みが印刷要求の正常終了を示していれば，バッファは“空き”という印がつけられる．(このバッファは IOSWITCH ポインタで指定されている.) PRINTINT は次にもう一方のバッファが一杯かどうかを調べる．一杯であれば，SIOSTART を呼出して入出力が起動され，制御は割込まれたプログラムに返却される．空であれば印刷すべきものは何もないので，IOSWITCH が“活動中でない”と設定され，制御は割込まれたプログラムに返却される．

入出力割込みが発生したとき，CPU は割込みを待って待合せ状態となっていることがある (PRINT ルーチン参照)．LPSW X'38' が実行されると CPU は再び待合せ状態となる．それで入出力用の旧 PSW の待合せ状態ビットをオフとしてから LPSW X'38' が実行される．図 2.32 に PRINTINT の流れ図を示す．

2.5.2.8 考察と演習問題

このプログラムはひとつの簡単な例題であるが，読者は入出力と割込みの微妙な関係をよく理解してほしい．とくに読者は次の問に答えられるようになっておいてほしい：

1. 毎秒 1,000 行の IBM 1403 型印刷機の場合には 1 行を約 60 ミリ秒で印刷できる．MAIN プログラムが 1 行印刷する時間はどれだけか．すなわち，PRINT を呼出す時間間隔は幾らか．とくに，60 ミリ秒より長

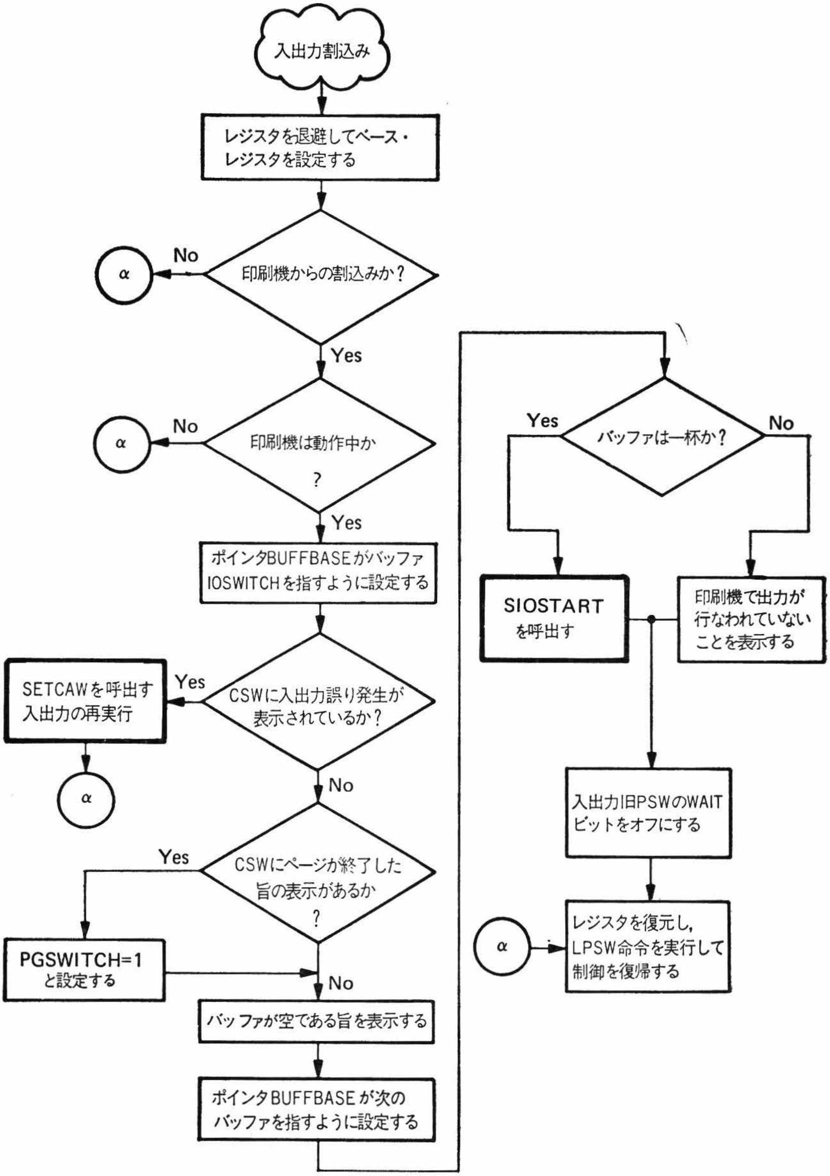


図 2.32　印刷機のため割込みハンドラ

いか短いか．簡単な命令（ロード，加算など）は1マイクロ秒，乗算命令は10マイクロ秒かかるものとする．

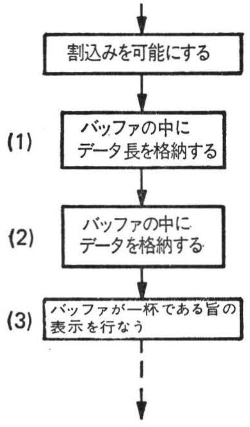
2. a．ダブル・バッファリングの技法の利点はなにか（問1に関連して答えよ）．

 b．単一バッファリングの技法の利点はなにか．

3. a．3重バッファリングにするにはプログラムをどのように変更すればよいか（この問題は簡単である．そうでなければ考え方が悪いのである）．

 b．3重バッファリングの利点はなにか．

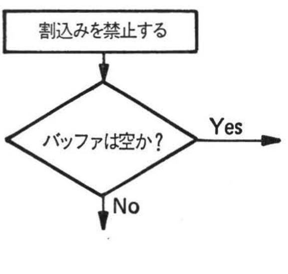
4. PRINT サブルーチンでは次の順で事象が起こった（図2.30参照）．



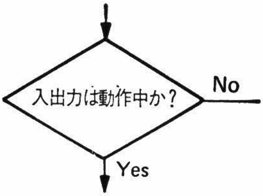
 a．この順を，たとえば(3)，(2)，(1)の順に変更すれば顕著な相違がみられるであろうか．（割込みはこれらの事象の間に発生することを思い出せ．）

 b．(2)，(3)，(1)の順にすればどうなるか．

5. バッファが空かどうかを調べる前に PRINT で割込みを不能にする理由は何か．



6. 入出力が活動中か否かを調べる前に割込みを不能にしなくてもよい理由は何か.



7. SIOSTART の中で割込みを不能にしなくてもよい理由は何か.

問4～問7はオペレーティング・システムにおいて重要な同期の問題を示している．読者は時間をかけて答えを考えてみてほしい．これらの問題は処理装置管理および装置管理の章で述べる.

2.5.2.9　考察と演習問題の解（1および4）

1. MAIN から PRINT を呼ぶ時間の間隔は発生される乱数によって異なる．この時間の近似値は

$$T_j = R_{j-1} \times R_{j-2} \times 14\ (\text{マイクロ秒}) + K$$

ループの所要時間は前述の条件では約14マイクロ秒である．K は残りの命令の分で100マイクロ秒以下である．R_{j-1} および R_{j-2} は最後に発生された2つの乱数である．積 $R_{j-1} \times R_{j-2}$ は1～10,000の値をとる．したがって T は100マイクロ秒から140ミリ秒の間の値をとる．とくに，この値は60ミリ秒よりも小さいことも大きいこともある．このように時間が変化することは，入出力要求の間隔が処理すべき入力によって異なる実際のプログラム（アセンブラやコンパイラなど）の場合には普通のことである.

4. 問の手順の任意の時点で割込みが発生して PRINTINT サブルーチンに制御が移り得る．もし順序を逆にして(3)，(2)，(1)として，(3)の直後に割込みが発生したとすると，図2.33に示したようなことが起こる.

このような場合には PRINTINT は情報がバッファに入る前に出力を開始してしまうので，以前からバッファに入っているすでに印刷ずみの内容を印刷するかもしれない．再び PRINT に制御が戻るとバッファの中にデータを移すので，印刷される行は，印刷出力と PRINT ルーチンの相対的な速度の関係によって，古い行と新しい行とが混合した内容に

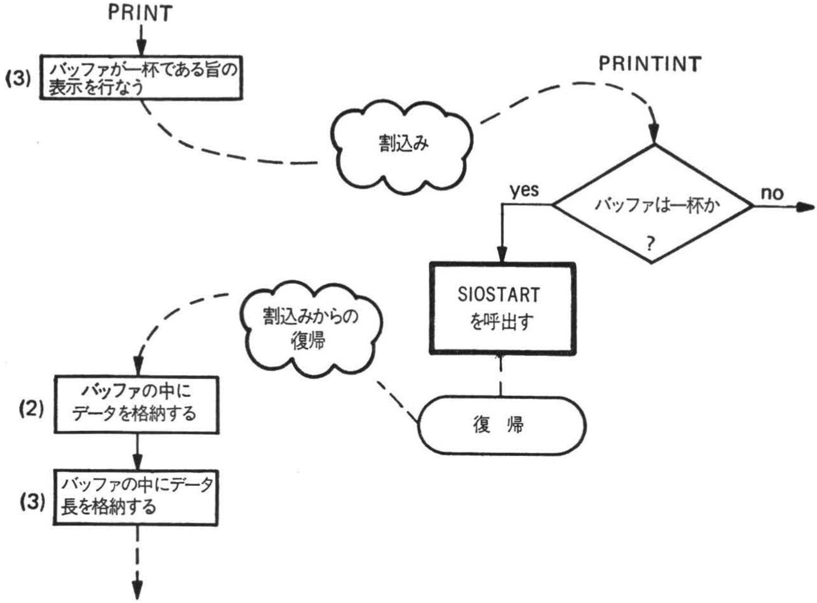


図 2.33　割込み処理の手順

なることがある．

順序が(3)，(2)，(1)で(2)の後で割込みが起こったらどうなるか考えてみよ．この場合も誤りが起こるか．

ここで述べた問題を解決するには2つの解決法がある．1つの方法は“安全な順序”を決定することである［(1)，(2)，(3)の順序は安全であることを確かめてみよ．すなわち，任意の時点で割込みが発生しても誤りとはならない］．他の方法はこの手順の間割込みを不能にすることである（たとえば，SSM=X′00′）．後の方法は，余分で不必要な“割込み不能状態”の使用が行なわれたり，とくに，その手順が長く時間のかかるときにはシステムの速応性の能力を著しくそこねるので，一般には推奨できない．一般的な経験則としては，絶対に必要となるまでは割込みを不能にしてはならない．

2.6 要　約

本章はオペレーティング・システムの設計と解析の1つの問題として，入出力（装置管理）および CPU の割込み処理（装置管理）について論じた．処理能率と利用効率を高める技法についても示した．これらのことは実際には通常のプログラマの仕事ではなくオペレーティング・システムの責任である．オペレーティング・システムは，最大のシステム仕事量や最高の優先順位などの一定の目標にしたがってそれに適するように，入出力や CPU のスケジュールを行なう．

本章では最近の計算機構成の具体的な例として機械語，入出力プログラミング，および割込みのハードウェアを示し，オペレーティング・システムで管理されるべき資源を理解し，認識する基礎を与えた．以下の章では，これらの資源を効率よく管理するために必要な技法と原理を理論的な順序で述べて行く．

演 習 問 題

2.1

1. 入出力チャネルとは何か．計算機システムでこれが果たす機能は何か．
2. CPU はチャネルとどのようにして通信するか．チャネルはどのように CPU に情報を返却するのか．
3. プログラム状態語（PSW）とチャネル状態語（CSW）の機能は何か．類似する点は何か．
4. チャネル番地語（CAW）は何のために使用されるのか．CPU でこれに対応するプログラム番地語を必要としない理由は何か．
5. 入出力命令（TIO, SIO など）とチャネル・コマンド語の相違点は何か．

2.2　IBM 370 で次の CCW が実行されたら何が起こるか．

CCW X'09', LINE, X'20', 120

2.3　IBM 370 の PSW，記憶装置，およびレジスタが次の値をとっているとき何が起こるか．

PSW - X'00A00000000EA634'
レジスタ 15 - X'000EA600'

番地(16進数)	値(16進数)
0EA634	D2030048F0409C00000E
0EA640	A00EA648
0EA648	8B00000060000001
0EA650	090EA6602000000D
0EA660	E8D6E440C1D9C540E6D9D6D5C7

2.4 次の用語を簡潔に定義して説明せよ．

1. チャネル
2. SIO
3. CSW
4. CAW
5. CCW
6. セレクタ・チャネル
7. マルチプレクサ・チャネル
8. システム・マスク

2.5

1. 次の CCW がチャネルで実行され，IBM 3211 型印刷機が選択されていたとしたら何が起こるか．

```
       CCW    X'09',DATA,0,3
DATA   DC     C'WIN'
```

2. IBM 3210 型遠隔端末装置がチャネルで選択されたら何が起こるか．IBM 3505 型カード読取り機の場合はどうか．

2.6 図 2.34 に示したのは入出力命令とチャネル・コマンド語を含んだ IBM 370 のアセンブラ語のルーチンである．

チャネル X'00E' が IBM 3505 型カード読取り機に，X'00B' が IBM 3211 型印刷機に接続されているとする．

1. このプログラムが何をするか簡単に述べよ．
2. 8行，9行，14 行および 15 行の TIO および BC 命令の機能は何か．他の2つの SIO 命令の前に同様な命令が使用されていない理由は何か．

```
文番号          原始文

 1 WALDO       START        0
 2             BALP.        15,0
 3             USING        *,15
 4 CAW         FOU          72
 5 LOOP        MVC          CAW(4),=A(CHAN1)
 6             SIO          X'00E'
 7             MVC          CAW(4),=A(CHAN2)
 8 TEST1       TIO          X'00R'
 9             BC           7,TEST1
10             SIO          X'00B'
11 TEST2       TIO          X'00E'
12             BC           7,TEST2
               .
               .
               .
               .
               .            BUFFER1
               .            の中のデータの処理および修飾
               .
13             MVC          CAW(4),=A(CHAN4)
14             TIO          X'00B'
15             BC           7,TEST3
16             SIO          X'00B'
17             MVC          CAW(4),=A(CHAN3)
18             SIQ          X'00E'
19             TIO          X'00E'
20             BC           7,TEST4
               .
               .
               .
               .
               .            BUFFER2
               .            の中のデータの処理および修飾
               .
21             B            LOOP
22             BR           14
23 CHAN1       CCW          X'02',BUFFER1,X'00',80
24 CHAN2       CCW          X'09',HEADER,X'A0',13
25             CCW          X'00',BUFFER2,X'40',13
26             CCW          X'11',TRAILER,X'20',16
27             CCW          X'02',BUFFER2,X'00',80
28             CCW          X'00',HEADER,X'A0',13
29             CCW          X'00',BUFFER1,X'40',13
30             CCW          X'11',TRAILER,X'20',16
31 HEADER      DC           C'START MESSAGE'
32 TRAILER     DC           C'MESSAGE FINISHED'
33 BUFFER1     DS           CL132
34 BUFFER2     DC           CL132'INITIAL MESSAGE'
35             END
```

図 2.34　IBM 370 のアセンブラ語ルーチンの一部

3. このプログラムのようにバッファを1つだけ使用すると領域が節約できる．バッファを2つ使用することの利点は何か．3つ以上のバッファを使用するともっと利益があるか．

4. WALDO を実行している間 120_{10} 番地は次の値（16 進数）を納めている：

0004 0000 0F0A 8248

a．WALDO の実行中に，このプログラムで起動した入出力動作の完了を通知する入出力割込みが発生したとする．この割込みを処理するために制御はどの番地に移されるか．

b．割込みルーチンの実行中にマスクされる割込みは何か．どの割込みは可能とされているか．

c．割込みルーチンの実行中に許される記憶保護キーはどんなものか．

2.7 通常，割込みの制御には何種類かの方法がある．IBM 370 には3種類の制御法がある．

無　視——割込みを無視することを指定する（すなわち，割込みは全く起こらない）．

延　期——割込みの発生を延期させる（行列に加える）ことを指定する．マスクが外されると割込みが起こる．

無条件——割込みを無視したり，延期させたりできない．

次に示す割込みに対してはどのような制御が適当か説明せよ．

1. スーパバイザ呼出し（SVC）
2. 保護侵害（プログラム割込み）
3. 入出力完了
4. 固定小数点あふれ（プログラム割込み）

2.8 割込みの構造——マスク

1. 割込みのマスクは何のために用いられるか．
2. 割込みをマスクするとその割込みはどうなるか．
3. ある種の割込みはマスクすることができない．次の割込みがマスク可能か不能かを理由と共に述べよ．

a．外部割込み

b．装置 X′00E′ よりの入出力割込み

c．例外的処理割込み

b．特権的処理割込み

e．保護割込み（用心）

f．番地指定割込み

g．固定小数点あふれ割込み

4.　X'78' 番地から開始すると
X'FF0C 000F 0E86 A8'
が出てくる．どのような問題が起こるであろうか．

2.9　次のものの相違点および類似点を論じよ：
PSW の割込み符号部（ビット 16～31）
PSW の条件符号部（ビット 34～35）
CSW の状態部（ビット 32～47）

2.10 PSW のシステム・マスクとプログラム・マスクの相違点は何か．

2.11 図2.22の文30でレジスタ15が絶対番地54に格納される理由は何か．
文 33　STM 0,15 を使用しない理由は何か．

2.12 入出力の初期化プログラムは普通次のようなコードを有している．

```
         ...
         SSM      OFF              すべての割込みを解消する
         TIO      XXX              装置が動作可能となるまでループする
         BNZ      *-4
         LA       1,CCWLIST        CAWを設定する
         ST       1,CAW
         SIO      XXX              入出力を起動する
         BNZ      ERROR
         LPSW     WAIT             入出力完了を待合せる
         ...
WAIT     DC       X'FF02 0000 0000 0000'
         ...
```

1.　SSM（システム・マスク設定）命令が必要な理由は何か．すなわち，なぜ上の手順で割込みを不能にしなければならないのか．

2.　どのような場合に SSM 命令を取除いても安全か．すなわち，割込みを可能として上の手順が安全に実行されるのはどのようなときか．

3.　WAIT が次のように変更されたら都合の悪いことは何か．

WAIT　DC　X'0002 0000 0000 0000'

2.13

1. 7つのチャネルに幾つまで入出力装置を接続して番地づけすることが可能か.
2. 読者は TWA 会社のシステム・プログラマとしてオンラインの座席予約システムを開発している. このシステムを IBM 360 で稼働させ, システムに 10,000 台の端末を接続することを望んでいる. 基本的なシステム構成を変更せずに, 基本の IBM 360 に接続すべき新装置を開発せよ.

2.14 次の文章は正しいか. その理由は何か.

"プログラム割込みが発生すると CPU は自動的に 104 (10 進) 番地に制御を移す."

2.15 IBM 370 には5種の割込みがある:

1. 外　部
2. スーパバイザ呼出し
3. プログラム
4. 機械検査
5. 入力／出力

計算機によっては, 割込みは同期と非同期の2種に分類されている:

1. 同期および非同期割込みの意味を定義せよ.
2. IBM 370 の割込みをこれらの2種に分類してみよ.

2.16

1. 割込みの優先順位の意味は何か. IBM 370 には5種の割込みがある. これらを優先順位の順に並べよ. またこれらの割込み原因を述べよ. 各種の割込みについて設計者がなぜそのような優先順位を定めたと思うか述べよ. もっとよい順位の定め方があるか.
2. 5種類の割込みを設ける理由は何か. 他の種類を追加する理由があるか. あれば説明せよ. あるいは種類を減らす理由があるか. あれば説明せよ. IBM 370 の割込みの種類はこれで十分か.

2.17

1. 割込みをマスクするのはどのような効果があるか. マスクはどのよう

にしてかけるのか．

2.　5種類の割込み（入出力，プログラム，機械検査など）でマスクできるのはどれか．

3.　次に示す割込みをマスクしたい場合の例をあげよ．それぞれの場合に割込みを禁止したい理由を説明せよ．

a．固定小数点あふれの例外状態

b．すべての入出力割込み

c．特定のチャネルからの入出力割込み

4.　割込みが発生するとPSWに何が起こるか．割込みの処理を終えたときどのようにして元に復帰するか．

5.　プログラム割込みの処理ルーチンを実行中に固定小数点あふれの状態が発生したら何が起こるか．このような状態を回復するにはどのようにすればよいか．これを防ぐにはどのようにすればよいか．

2.18 次の割込みはどのようなときに起こるか．

1.　チャネル動作完了

2.　プログラム制御割込み

3.　装置動作完了

2.19 IBM 370 でチャネル動作完了と装置動作完了が別になっている理由は何か（どのような利益があるか考えよ）．

2.20 IBM 370 ではロード命令に対応した格納命令が設けられている．しかし LPSW（ロード PSW）には対応した命令がない．

1.　PSW 格納命令をシミュレートすることができるか．できるとすればどのようにするのか．

2.　オペレーティング・システムはなぜ使用中の PSW の内容を調べる必要があるのか．

2.21 図 2.18 の文4ではなぜシステム・マスクが設定されたのか．

2.22 プログラム割込みが発生したときメッセージを印刷する割込み処理ルーチンがあるとする．割込みが発生してその処理を終え，誤りを発生した命令を無視して元のプログラムに復帰するものとする．56 番地には 16 進で次の内容が格納されている：

FFA1 0004 A000 B204

1. 何番地に復帰すべきか.
2. 割込み発生前に実行していた命令は何番地か.
3. どのような誤りでこの割込みが起こったのか.
4. この割込みを起こした命令は記憶装置のある番地αを参照しようとしていた. αが含まれているブロックに割付けられていなかった記憶保護キーの値の1つは何か.

2.23 図 2.35 のプログラム HARRY, INT, および CCW を考える.

1. X'00E' に IBM 3211型印刷機が接続されているとし, これらのプログラムがロードされ HARRY から実行開始されたとすると何が起こるか. 主ルーチン HARRY は BALR 14, 15命令によって呼出されると仮定する.
2. HARRY は次の命令列によって2回呼出されるとする.

```
L       15,=A (HARRY)
BALR    14,15
L       15,=A (HARRY)
BALR    14,15
```

3つのルーチンをたどってみて何が起こるか確かめたうえで次の問に答えよ.

"HARRY の第6および第7行のコードが無限ループでない理由は何か".

3. HARRY の第6〜7行および INT の第7〜9行で実行される機能は何か.
4. 上の第3行と同じことを行なうにはどの命令を用いればよいか. どのように変更したらよいか示せ.

2.24

1. CCW よりも便利な入出力プログラミング用の記号言語を定義せよ.
2. IBM は, 入出力の処理のために利用者が "入出力用の言語" でプログラムを書く代わりにシステム・マクロを使用するようにしている. アセンブラを用いるよりこの方法が優れている点を示せ.

2.25 IBM 370 ではオペレーティング・システムの保護のために2つのハー

```
文番号          原始文

 1  HARRY       START
 2              ENTRY FLAG
 3              EXTRN INT,CCW
 4              BALR   15,0
 5              USING *,15
 6  LOOP        CLC    FLAG,=F'0'
 7              UNZ    LOOP
 8              LA     1,1
 9              ST     1,FLAG
10              MVC    NIOPSW (8),IOPSW
11              MVC    CAW (4), =A (CCW)
12              SIO    X'00E'
13              BR     14
14  NIOPSW      EQU    120
15  CAW         EQU    72

16  IOPSW       DC     A (0,INT)
17  FLAG        DC     F'0'
18              END
```

```
文番号          原始文

 1  INT         START
 2              EXTRN FLAG
 3              ST     15,8
 4              BALR   15,0
 5              USING  *,15
 6              STM    1,2,SAVE
 7              SR     1,1
 8              L      2,=A (FLAG)
 9              ST     1,0 (2)
10              LM     1,2,SAVE
11              L      15,8
12  DONE        LPSW   OIOPSW
13  OIOPSW      EQU    56
14  SAVE        DS     2F
15              END
```

```
文番号          原始文

 1  CCW         START
 2              CCW    X'09',FIRST,X'80',14
 3              CCW    X'00',ANS,X'80',4
 4              CCW    X'00',SECOND,X'00',6
 5  ANS         DC     C'6802  LOSES.'
 6  SECOND      DC     C'  WINS.'
 7  FIRST       DC     C'THE ANSWER IS '
 8              END
```

図 2.35　HARRY, INT, CCW プログラム

ドウェア機能を利用している.

1. 記憶保護キー
2. スーパバイザ／プロブレム・モード――SSM, SIO などの命令の使用を制限する.

1. 記憶保護キーがどのように働くかを述べよ.
2. スーパバイザ／プログラム・モードがどのように働くかを述べよ.
3. 記憶保護機構がすべての保護の要求を処理できるようにするためには，ハードウェアをどのように修飾すればよいか（すなわち，スーパバイザ／プロブレム・モードの削除).（**ヒント**：使用中の PSW に記憶装置の番地を割付ける.）説明せよ.
4. スーパバイザ／プロブレム・モードの機構を拡張して記憶保護を処理するようにできるか. 理由および方法を示せ.

2.26

1. 保護キーと保護ロックの相違点は何か.
2. PSW の保護キーが記憶ブロックにつけられたロックに合えばそのブロックに対してどんな参照も許されるか.
3. 合わないときにはどうすれば参照できるか.
4. IBM 370 は 2 種類のロック――取出し，取出しと書込み――しか備えていない. 実行ロックの効果を IBM 370 でシミュレートすることができるか（すなわち，ブロックは読出せるが実行できない). それはどのようにするか.

2.27 IBM 370 のロックは 4 ビットである. したがって記憶装置は 16 人まで利用者に分割して割付けことができる. この 4 ビットを用いて 16 人以上の利用者に対して分割割付けができるか. それはどのようにするか.

2.28 図 2.24 のプログラムを書換えて第 2.3.7 節で述べたダブル・バッファリングを行なうようにせよ.

2.29 第 2.5.2 節の入出力バッファリングの例で示したルーチンをアセンブラ語で作成してみよ.

機械の問題と設計の問題[1]

2.30　“A 1,889” などの命令を IBM 360 で実行すると，正しい語境界を指定していないのでプログラム割込みが発生する．このような番地指定誤りを受理して訂正するような割込みルーチンを IBM 360 のために作成せよ．すなわち，IBM 360 を，整列していないデータ処理の可能な IBM 370 として働かせるようにせよ．

2.31　IBM 360 と IBM 370 はきわめてよく似た計算機であるが，IBM 370 には2，3機能が多い．IBM 370 の2進形式のデックを IBM 360 で実行しようとする．IBM 370 の MVCL（長い移動）命令以外はそのまま実行できるとする．

上の命令が実行されようとすると，IBM 360 ではプログラム割込みとして処理される．MVCL 命令を“模擬”してから次の命令の実行を再開するプログラム割込み処理ルーチンを作成すると，上のプログラムを実行することができる．このような割込みハンドラを作成せよ．

ただし，プログラム割込みは上述の場合だけで起こるとし，プログラム割込み用新 PSW は次の値に設定されているとする：

```
PINPSW   DC A(0,MVCLSIM)
```

ここで MVCLSIM は割込み処理ルーチンの名前とする．

---

1) 本節の問題は設計の問題としてでも機械の問題としてでも使用できる．設計の問題のとき答えは流れ図と設計文書が必要である．機械の問題のとき答えは実際のプログラムとして作成する必要がある．たいていの計算機システムでは通常の利用者が入出力プログラムや割込みプログラムを作成して実行することができないので，作成したプログラムを組込むことは困難と考えられる．しかし次のようにして組込みを行なうことが可能であろう．

1　独立の IBM 370 を使用しているとき，これらのプログラムを直接組込むことが可能である（たとえば，IBM 370/135 やスーパバイザ状態で実行可能の計算機）．

2　大型の IBM 370 でバッチ処理を行なっている場合には，裸の計算機に触れることはできないが，シミュレータを使用することができる．シミュレータは IBM 370 をシミュレートして“仮想的な IBM 370”を作り出す．このシミュレータは PL/I で書かれており，著者に申出てもらえば入手可能である．

3　他の計算機がある場合（たとえば PDP-8），たいていの問題はその機械の問題に変更することができる．

4　仮想計算機 370（VM/370）に接続されている端末装置がある場合には，この仮想計算機に組込むことができる．

MVCL 命令

0E	R1	R2

R1 および R2 はそれぞれ汎用レジスタの対を指定する．たとえば，0E28 という命令が実行されたとして，レジスタには次の値が入っていたとする．

レジスタ2 = 00001800,　レジスタ3 = 00001000
レジスタ8 = 00043700,　レジスタ9 = 00001000

このとき上の命令を実行すると，4096 ($1,000_{16}$) バイトが $43,700_{16}$ 番地から $1,800_{16}$ 番地に移動される．（註：実際の MVCL 命令はもう少し複雑である．本問では例に示したような2つの部分の長さが等しい場合のみを考える．）

2.32 IBM 370 計算機の割込みの1つにプログラム割込みがある．この割込みが起こったとき，ハードウェアは，“この命令の実行が異状処理を呼出した”旨信号を送る．たとえば，固定小数点加算 (A, AR) で結果を表わすのに 32 ビット以上必要となるとプログラム割込みが起こる．使用中の PSW の割込み符号が，固定小数点あふれが起こったことを示すように設定され (0008)，PSW 全体がプログラム用の旧 PSW (28_{16} 番地) に格納される．最後に，プログラム用の新 PSW (66_{16} 番地) が PSW として設定され，次の命令（番地はプログラム用の新 PSW の下位3バイトに納められている）が実行される．すなわち，制御はプログラム割込みハンドラに移される．

この機械の問題では割込みハンドラを作成する．

問題は3段階に分かれてむずかしくなっている．解答には次の2つの項目を含めること：

1. 各行に註釈をつけたアセンブラ語のプログラム．
2. 1～2ページの範囲でプログラムの内容，プログラム検査の結果を記述せよ．プログラムが正しく働かなかったときはその理由を述べて提出してもよい．

1. 次のことを行なうプログラム割込みハンドラを書け．

a．固定小数点あふれによるプログラム割込みだけを処理する．他の種

類のプログラム割込みは，（割込みを起こした命令の次の命令に復帰して）無視する．

b．割込みを起こした命令を解析し，R1部で指定されたレジスタに32ビットで表現できる最大正数を設定する．ここではAおよびAR命令のみ処理すればよい．（最大正数を設定しておくことは場合によっては合理的なことである．）

c．もちろん割込まれたプログラムにとって条件が不変となるようにレジスタ（R1部で指定されたレジスタを除く）を復元した後で復帰しなくてはならない．

d．割込みを起こした命令の次の命令に復帰する．

e．プログラムの働きを証明するための証拠を作成しておかねばならない（たとえばTRACE出力）．

2. 次のいずれかを選択せよ．

a．プログラム割込みハンドラを変更して，プログラム割込みハンドラの中でスーパバイザ呼出し割込みを起こすことができるようにせよ．すなわち，1つの割込みを処理中に割込みが発生して他の割込みハンドラを実行してもよいようにする．SVC割込みハンドラを作成しなければならない．SVCハンドラで，IPLボタンが押されて以来何回SVCが実行されたかを数えておくと面白い．SVCの回数を数えたら次の命令に復帰する（すなわち，SVC割込みハンドラは単にSVCの回数を数え制御を返却する）．

b．プログラム割込みハンドラの設計を変更して，主プログラムはユーザ・モードで，割込みハンドラは特権モードでそれぞれ実行するようにせよ．主プログラムが実際にユーザ・モードで実行しているかどうかを検査するために主プログラム内で特権命令の実行を試みてみよ．この結果プログラム割込みが発生するであろう．プログラム割込みハンドラでは“特権処理”割込みであるかどうかを調べ，何回特権命令がユーザ・モードで実行されようとしたかを数えよ．

3. 固定小数点あふれが起こったとき，印刷機で次のメッセージを印刷せよ：

“FIXED POINT OVERFLOW AT XXXXXX”

ここで XXXXXX は固定小数点あふれの割込みを起こした命令の絶対番地の16進表現である．この処理のコードはプログラム割込みハンドラの内に含まれなければならない．

第3章

記憶管理

オペレーティング・システムの記憶管理モジュールは主記憶の管理を行なう．**主記憶**とは処理装置が直接，命令やデータを参照する記憶装置のことをいう（図 1.1 参照）．主記憶は，多年にわたって使用されてきたフェライトの磁気コアにちなんで，しばしば**コア**とも呼ばれる．記憶管理は次の4つの機能を果たす：

1. 主記憶の各記憶場所の状態，すなわち，各記憶場所が割付けられているかいないか（空いている）を把握する．
2. 記憶領域の割付け方針，すなわち，どのプロセスに，いつ，どの領域を，どれだけ与えるかを決定する．主記憶が1つ以上のプロセスから同時に共用されているときは，記憶管理はどのプロセスの要求を満足させるかを決定しなければならない．
3. 割付けの技法——記憶領域を割付けることが決まると，特定の領域が選択され，割付け管理のための情報が更新される．
4. 領域解放の技法と方針——記憶領域の返却を処理する．プロセスは先に割付けられた記憶領域を自ら解放することもあるし，記憶管理が領域解放の方針に従って領域の返却を要求することもある．領域解放後には状態情報を更新しておかなければならない．

記憶管理で採用される方針は，記憶管理モジュールを小さく単純にしたいという要求とか，利用者が使いやすくしたりシステム効率を向上させたりする必要性などによって左右される．本章では各種の方針や技法を示す．あるものは記憶装置がきわめて効率よく使用できる．あるものは非常に複雑になり，ハードウェアの経費がかさみ，オーバヘッドが増すこともあるが利用者にとってより使いやすい．

ここで述べる技法を次に示す：

1. 単一連続記憶割付け
2. 分割記憶割付け
3. 再配置型分割記憶割付け
4. ページング記憶割付け
5. デマンド・ページング記憶割付け
6. セグメンテーションによる記憶割付け
7. セグメンテーション・デマンド・ページング記憶割付け
8. その他の記憶管理方式

拡張した計算機という観点（図 1.9）からすると，記憶管理モジュールは，通常，オペレーティング・システムの最高位のレベル（すなわち，これらのモジュールは，通常，他のレベルから呼出される.）にも最低位のレベルにも属していない．しかし現状では，記憶管理はたいていのオペレーティング・システムにおける顕著な特徴であるので，記憶管理を最初に論じることにする．SDS 940 と OS 360/MVT オペレーティング・システムとを明確に区別する特徴は記憶管理である（ページングと分割割付け）．

本章では，ジョブ処理に関連した記憶管理の役割りを強調する．しかし，計算機システムの中では，記憶管理に対して多くの要求が存在する．これは卸売りと小売りのような関係の場合もあるし，より効率がよいとか，ある種の条件により適合しているという理由で別のアルゴリズムが採用される場合もある．

本章で示した技法はたいていの記憶管理の応用を含んでいる．たとえば，あるオペレーティング・システムやその中での利用者に対しては，ジョブ処理の記憶管理はデマンド・ページング割付け法に基づいて行なわれている．そのような環境で利用者プロセスは分割割付けの技法に基づいて，割付けられた記憶領域を個々のデータ領域に再割付けする．同時に，オペレーティング・システムは自分の使用する動的なシステムの表を処理するのに再配置型分割割付け法を使用することもある．このような理由によって，オペレーティング・システムは別個の目的のために使用可能な多様な記憶管理機能を備えることもある．

解析をできるだけ単純とするために，本書ではジョブ処理における記憶管理の要求に焦点を合せることにする．他の場合についても同様にして解析を行なうことができる．第7章の例題のオペレーティング・システムでは幾つかの異なった場合について説明している．

記憶管理がシステム効率に及ぼす影響を説明するために，本章では多重プロ

プラミングの概念を紹介する．第4章においては多重プログラミングについてさらに進んだ議論を行なう．

本章で論じる記憶管理の方針と技法の解析は，それぞれ次の4つの節に分かれる：

1. 採用した解析法と原理の概説
2. 必要とされたり望ましいと考えられる特殊なハードウェアの記述
3. 必要とされるソフトウェアのアルゴリズムや処理の記述
4. この方策の利点と欠点の考察

3.1 単一連続割付け

単一連続割付けは特別なハードウェア機能を必要としない単純な記憶管理方式である．この方式は，通常，バッチ•オペレーティング•システムを持つ小規模な単独の計算機（あるいはミニコンピュータ）で使用される．この例としては，IBM OS/360基本制御プログラム，IBM 1130ディスク・モニタ・システム，IBM 7094 FORTRANモニタ•システムなどがある．このようなシステムでは多重プログラミングは行なわれておらず，利用者，ジョブ，ジョブ・ステップ，およびプロセスは1対1に対応する．本章では簡単のためにこの1対1の対応を仮定することとし，“ジョブ”という言葉のみを使用する．記憶領域は図3.1に示したようにジョブに割付けられる．

記憶領域は概念的に3つの連続した領域に分割されている．1つの部分は恒久的にオペレーティング・システムに割付けられている．残りの領域は処理中の単一のジョブで使用可能（割付けられている）である．ジョブは，実際には割付けられた領域の一部分しか使用せず，割付けられたが未使用のままの領域

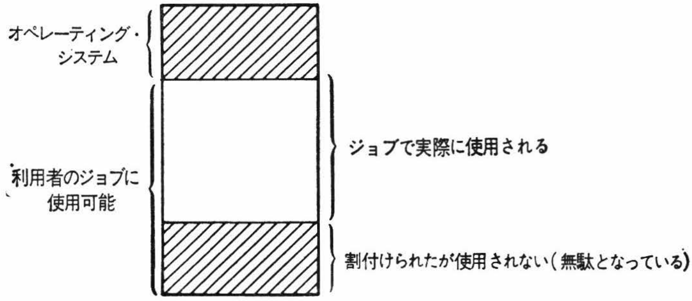


図 3.1 単一連続割付け

を残している．たとえば，256K バイトの記憶領域があり，簡単なオペレーティング・システムが 32K バイトを必要としたとすると，224K バイトを利用者ジョブに割付ける．あるジョブが 64K バイトだけ必要としたとき，160K バイト（全記憶領域の50%以上）は使用されない．この使用されない記憶領域は，224K バイトを全部必要とするジョブが存在することもあるので，計算機メーカーに返却することはできない．

記憶管理の4つの機能がどのように処理されるかを調べてみると，この方式の主たる利点である単純性に気がつくであろう：

1. 記憶装置の状態を把握すること——全部の領域が1つのジョブに割付けられる．
2. 記憶割付けの方針を決定すること——スケジュールされたときジョブはすべての記憶領域を得る．
3. 記憶領域の割付け——領域はすべてそのジョブに割付けられる．
4. 記憶領域の返却——ジョブが終了したとき，すべての記憶領域は空きの状態に復帰する．

3.1.1 ハードウェア

連続割付けには特別なハードウェアが必要でない．時として，利用者プログラムが事故や故意によってオペレーティング・システムを変更したりせぬように基本的な保護機構のハードウェアの設置が望まれる．この機構は CPU の限界レジスタと スーパバイザ/ユーザ・モードとから構成される．限界レジスタは保護すべき領域（ここにオペレーティング・システムが含まれる）の番地を格納している．CPU がユーザ・モードであるとき，記憶装置の参照が行なわれる度に保護領域へのアクセスでないことを確認する検査がハードウェアで行なわれる．もしそのようなアクセスが試みられたならば，割込みが発生して制御はオペレーティング・システムに移される．スーパバイザ・モードのときオペレーティング・システムは保護領域を参照したり，限界レジスタの内容を変更する特権命令を実行したりすることができる．そのほかに IBM 370 のロックとキーによる保護機構なども同様な効果を達成するために使用することができる．モードは，制御が（通常は割込みによって）オペレーティング・システムに移されたときに限ってユーザ・モードからスーパバイザ・モードに変更することができる．

3.1.2 ソフトウェア

単一連続割付けの流れ図は 図 3.2 に示されている．このアルゴリズムは，処理装置管理のジョブ・スケジューラがジョブを実行させるようにスケジュールしようとするときに呼出される．このアルゴリズムは他のジョブが記憶装置を使用中でないときに限って呼出される．

3.1.3 利 点

この方式の主たる利点は単純性にある．この方式のオペレーティング・システムは，もっと複雑なオペレーティング・システムが256Kバイト以上もの領域を必要とするのに比べ，全体でわずか1Kバイトあればよい．他の利点はこのようなシステムを理解したり使用したりするのに熟練が不要であることである．

3.1.4 欠 点

この方式の主たる欠点は記憶領域が完全に利用されていないということである．3つの点で効率が悪い．第1，記憶領域のある部分は全く使用されない

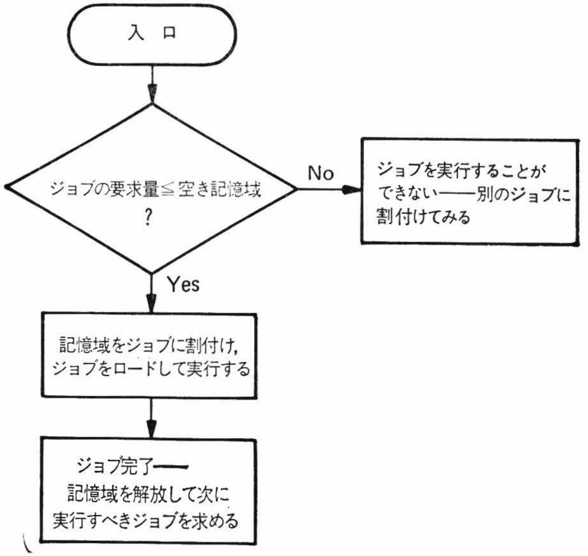


図 3.2 単一連続割付けの流れ図

(図3.1の無駄な部分を見よ). 第2, チャネルを備えたシステムでは, たいていの場合, ジョブが入出力要求を起動してチャネルが動作を完了するのを待合せているようなときなど, CPU で記憶装置が全く使用されないような期間が存在する. CPU が待合せ中には利用者のプログラムを格納した記憶領域は利用されていない. 何パーセント位の時間 CPU は入出力を待合せるであろうか. プログラムによって異なっているが, 単一連続割付けが行なわれると IBM 360/65 での典型的なジョブは 60〜70% の時間を待合せている. したがって, 記憶装置も処理装置も共に利用効率が低い. 第3, 利用者の番地空間を納めている記憶領域のある部分は, アクセスされることのない情報を格納している. たとえば, ジョブを構成しているプログラムやデータの集合には, しばしば, 実行されることのない誤り処理ルーチンが存在する. 記憶空間の中にこのようなルーチンまでロードしなければならない理由は何か.

他の欠点は, この方式は融通性に欠けることである. たとえば, 番地空間（プログラムの大きさ）は主記憶容量よりも小でなければならない. さもなくばジョブを実行することができない.

要約すると, 欠点は次のとおりである:

1. 記憶領域の利用率の低さ,
2. 処理装置の利用率の低さ（入出力待合せ),
3. 利用者のジョブは主記憶の利用可能な大きさに制限される（オーバーレイ機構の場合を除く).

3.2 多重プログラミング序論

単一連続記憶割付けに関する効率の悪さは, 固定の利用可能な資源と変化する必要資源との間の整合がうまく取れていないことに起因している. 計算機システムの物理的なハードウェア資源は, 比較的長い時間の間でのみ変化する. たとえば, 記憶装置を1単位（64K）追加したり削減したりすることなどの物理的な変更は通常1年に1回ないし2回行なわれるのにすぎない. ところが, 実行しようとするジョブが要求する資源はジョブごとに大きく変化する. すべてのジョブが全く同一の資源を要求するようにせよとプログラマに命令しようとすることは可能である. しかし, これは不可能ではないにしても極度に困難なことである. もっとうまい方法は, 2つ以上のジョブを同時に実行し, これらのジョブに資源を分配することである. この**多重プログラミング**と呼ばれる技法は, 2つ以上のプロセスの同時実行を行なうことである. 多重プログラミ

ングについては第4章（処理装置管理）において詳細に論じる．本節では，記憶管理の問題点を明らかにするために多重プログラミングの概念を簡単に紹介する．

3.2.1 多重プログラミングの例

図 3.3 は実行しようとする3つのジョブを示している．各ジョブは，計算（C_i）のために CPU 時間を，入出力（I_i）のためにチャネルを使用する必要がある．それに加えて，各ジョブの番地空間は記憶領域（M_i）を要求する．簡単のために，各ジョブは計算を行なっては入出力を行なうものであるとしてモデルを作っておく．このサイクルが連続して繰返される．記憶領域は 100K バイト使用可能であると仮定する．ジョブ1だけが実行されるとすると，30K バイトだけが使用され，残りの 70K バイトは無駄になっている．さらに，CPU 時間の $\frac{I_1}{C_1+I_1}$ が入出力を待合せて浪費されている．

他方，ジョブ1，2および3の番地空間を同時に主記憶の中に入れることもできる（30K＋50K＋20K＝100K バイト）．この方式では 100K バイトの主記憶を余すことなく利用する．処理装置管理は CPU をジョブ1に割付ける．計算 C_1 を行なった後で，I_1 が完了するのを待合せて CPU を停止する代わりに，処理装置管理は CPU をジョブ2に割付けることができる．同様にして，ジョブ2が I_2 を行なうようになれば，CPU はジョブ3に割付けられる．ジョブ3が I_3 を行なうようになり，I_1 がすでに完了していれば（すなわち，$I_1<C_2+C_3$），CPU は直ちにジョブ1に割付けることができる．このように，CPU は I_1 ではなく，$I_1-(C_2+C_3)$ だけ待合せればよい．多重プログラミングを行なうと，一時に1つのジョブの実行を行なっているときよりも CPU の利用率ははるかに高い．多くの場合に，多重プログラムを行なわずに1つのジョブを処理するのとほとんど同じ時間で2ないし3ジョブの処理を完了することが可

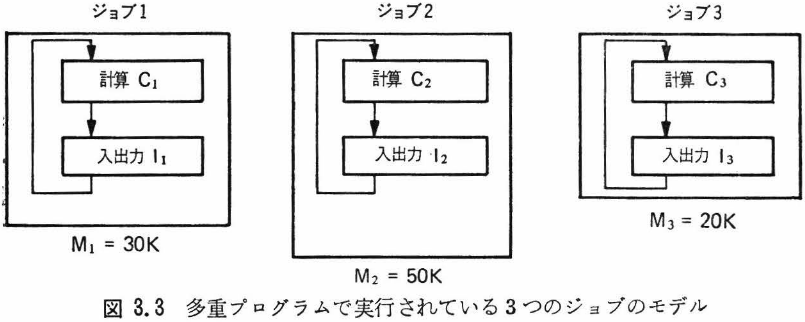


図 3.3 多重プログラムで実行されている3つのジョブのモデル

能である．

3.2.2 システム入出力待合せ率

現実の処理は図 3.3 のモデルのように単純ではない．計算と入出力は複雑に混ざり合っている． ジョブの入出力待合せ時間率 ω はこの関係を示すのに有用である．

$$\omega=\frac{\text{入出力待合せ時間の合計}}{\text{入出力と CPU 時間の合計}}$$

この値は，たいていの計算機では，多重プログラミングを行なわずにプロセスを実行してみて CPU が待合せ状態になっている時間の割合を調べると容易に測定することができる． 幾つかの研究の結果， IBM 360/65 や 370/158 などの中ないし大規模の計算機では，典型的なジョブで平均的に約 65% の入出力待合せ時間があることがわかっている．もちろん，個々のジョブはこの平均値から大きくずれていることもある．

$\omega=50\%$ であるジョブが 2 つあったとし，それらが多重プログラミングで実行されていたとすると，**実効的なシステム入出力待合せ率** ω' は 0 になるであろうか．図 3.3 のモデルで $C_1=I_1=C_2=I_2$ であることもある． このときには C_2 と I_1 を同時に実行し，C_1 と I_2 とを同時に実行することができる． しかしながら先にも述べたように，このモデルは単純化しすぎている．図 3.4 には計算と入出力の期間が混ざったプロセスの例が示されている．この例では C_{11}，C_{12}，C_{13} などの処理時間は等しいと仮定している．各ジョブの入出力待合せ率

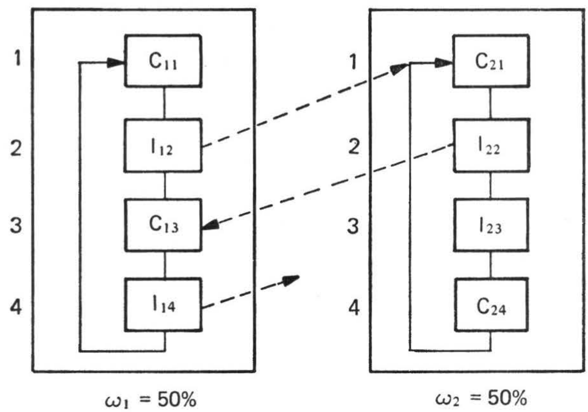


図 3.4 プロセス内での計算と入出力を示すもっと詳細なモデル

は50%である．これらのジョブを多重プログラミングすると，I_{12} と C_{21}，および I_{22} と C_{13} とを組合せることができる．しかし，ジョブ1が入出力 I_{14} を実行しようとするとき，ジョブ2も入出力 I_{23} を行なおうとしている．このようにして CPU は I_{14} あるいは I_{23} が完了するのを待合せている間アイドルとなる．

図3.4のモデルを一般化すると，確率論を応用して，実効的なシステム入出力待合せ率を推定することができる．システムは，すべてのプロセスが同時に入出力を要求しているときに限って待合せを行なわねばならない．多重プログラミングを行なっていて，各ジョブの入出力待合せ率を ω とすると，システム入出力待合せ率 ω' は次式で近似できる：

$$\omega' = \omega^n$$

例として，ω=50%のとき次の表が求められる：

プロセス数，N	システム入出力待合せ率，ω'
1	$(.50)^1 = 50\%$
2	$(.50)^2 = 25\%$
3	$(.50)^3 = 12.5\%$
4	$(.50)^4 = 6.3\%$
5	$(.50)^5 = 3.1\%$
6	$(.50)^6 = 1.6\%$

上の表はかなりよい近似ではあるが完全には正しくない．確率論の仮定が成立するためには，すべてのプロセスが各時刻に“1ステップ”ずつ進行する必要がある．n 台の処理装置と n 台のチャネルがあるときにもこの関係は成立する．ω' は n 台の処理装置がすべて同時にアイドルである確率を示す（n 枚の硬貨を投げてすべてが表向く確率）．マルチプレクサ・チャネルが複数台あるときなどは特に，n 個の入出力動作が同時に行なわれていることを仮定するのは道理のあることである．しかし，処理装置は1台だけであるから，この処理装置に仕事がたまって行く．この問題は**生死のマルコフ・プロセス**と呼ばれる確率の定理を用いて解くことができる．これについては第4章で論じる．この手法によってシステム入出力待合せ率はもっと正確に次式で近似できる：

$$\omega' = \frac{\left(\dfrac{\omega}{1-\omega}\right)^n}{n!\displaystyle\sum_{i=0}^{n}\frac{\left(\dfrac{\omega}{1-\omega}\right)^n}{i!}}$$

多重プログラムで実行されているジョブ数	各ジョブの入出力待合せ率 (ω)																		
1	5.0	10.0	15.0	20.0	25.0	30.0	35.0	40.0	45.0	50.0	55.0	60.0	65.0	70.0	75.0	80.0	85.0	90.0	95.0
2	0.1	0.6	1.3	2.4	4.0	6.0	8.6	11.8	15.5	20.0	25.2	31.0	37.6	45.0	52.9	61.5	70.7	80.2	90.0
3	0.0	0.0	0.1	0.2	0.4	0.9	1.5	2.5	4.1	6.3	9.3	13.4	18.9	25.9	34.6	45.1	57.2	70.6	85.1
4	0.0	0.0	0.0	0.0	0.0	0.1	0.2	0.4	0.8	1.5	2.8	4.8	8.1	13.1	20.6	31.1	44.7	61.4	80.2
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.3	0.7	1.4	2.9	5.8	11.0	19.9	33.6	52.5	75.3
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.4	0.9	2.2	5.2	11.7	24.1	44.1	70.4
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.2	0.7	2.2	6.3	16.3	36.2	65.7
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.2	0.8	3.0	10.4	28.9	60.9
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.3	1.3	6.1	22.4	56.3
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.5	3.4	16.8	51.7
11	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	1.7	12.1	47.2
12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.8	8.3	42.7
13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	5.4	38.5
14	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	3.4	34.3
15	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	2.0	30.3
16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.1	26.4
17	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6	22.8
18	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	19.4
19	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	16.3
20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	13.4

表 3.1 多重プログラミングにおけるシステム入出力待合せ率 (ω')

ω' の値は **表 3.1** に示されている．この式によると $\omega'=\omega^n$ よりも小さな値が得られる．その理由は，n 台の処理装置ではなく，1 台の処理装置ですべての仕事を実行しようとするからである．

3.2.3 多重プログラミングと記憶管理との関係

上述の式はただの近似にすぎない．重要なことは，一般に，入出力待合せ時間は，**多重プログラミングの多重**度，すなわち，多重プログラミングされるプロセスの数が増加すると減少するということである．以下に述べる記憶管理のアルゴリズムは，多重プログラミングの多重度を最大とするように記憶装置を割付けようとするものである．

3.3 分 割 割 付 け

分割割付けには種々の形態があるが，これは多重プログラミングを可能とするための最も簡単な記憶管理技法の 1 つである．主記憶はいくつかの**記憶領域**である**区画**に分割される．各区画は，図 3.5 に示したように，別々のジョブの番地空間を収容する．

この技法を用いて，記憶管理の 4 つの機能は次のように簡単に処理される：

1. 各区画の状態を把握する（たとえば，“使用中”/“未使用”，容量）．

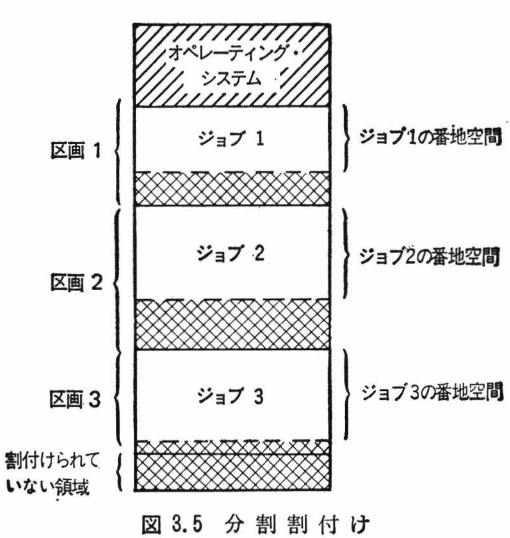


図 3.5 分 割 割 付 け

2. 記憶領域を使用するプロセスを決定する——これはジョブ・スケジューラによって処理される.
3. 割付け——利用可能の，十分な容量の区画が割付けられる.
4. 解　放——ジョブが完了したとき，その区画は“未使用”の印がつけられ，他に割付けることができる.

これらの機能の詳細についてはソフトウェアの節で示される.

3.3.1 ハードウェア

分割割付けのためには特別なハードウェアはほとんど必要でない. 1つのジョブが，事故にせよ悪意からにせよ，オペレーティング・システムや他のジョブを壊すことを防ぐために，記憶保護機構を備えていることが望ましい. これには種々の方法がある. 1つの方法は，使用する区画を囲む2つの限界レジスタを用いることである. このジョブが区画の外側の記憶領域を参照しようとすると保護割込みが発生する. この方式には次の2つの欠点がある：

1. 多重プログラミングによって処理装置が別のジョブに割付けられる度に限界レジスタを変更しなければならない.
2. この保護を入出力チャネルに拡張することは困難である. 区画1に入っているジョブが，チャネルに命じて区画3に入っているデータを読出させるかもしれない. 入出力チャネルごとに限界レジスタを設置することも可能であるが，マルチプレクサ・チャネルに対しては多数のレジスタが必要である. その代わりに，すべての入出力動作を起動するオペレーティング・システムがソフトウェアによってチャネル・プログラムを検査することもできる. しかし，この方法は時間のかかる処理である.

IBM 370 の保護機構はこれらの要求をかなりうまく取扱っている. 各区画には別々の保護キーが割付けられている（1～15). キー0はオペレーティング・システムで使用される. ハードウェアではキーは2Kバイトの記憶ブロックごとに与えられるので，区画は2Kの倍数である必要があり，1つの区画に含まれるすべてのロックは同じ値に設定しておかなければならない. 処理装置が他に割付けられたとき，新しいプログラム状態語の中の“キー”部は対応する区画の保護キーの値に設定される. 入出力チャネルも同様な保護キーの機構を使用している.

3.3.2 ソフトウェアのアルゴリズム

分割割付けの手法は，これから述べて行く記憶管理技法と同様に，多数の変形がある．一般的なものは**静的な分割**と**動的な分割**の2つである．

3.3.2.1 静的な分割

静的であるとは，記憶領域がジョブ処理に先立ってあらかじめ区画に分割されていることを意味する．これは IBM OS/360 の MFT（固定タスク数の多重プログラミング）で採用されている技法と同じである．区画はオペレータによって指定されたり，オペレーティング・システムに組込まれて定まっていたりする．静的な区画指定表の例を 図 3.6 に示す．

利用者の各ジョブ・ステップは必要とする記憶容量の最大値を指定しておく必要がある．十分な容量の区画が探されて割付けられる．割付けと解放を処理するソフトウェアのアルゴリズムは簡単なものである．

静的な割付け法は，特に，ジョブの容量と頻度がよくわかっているときに適する技法である．このような場合には，区画の容量が平均的なジョブの容量に合せて決定される．しかし，ジョブの容量や頻度が不明のときや分散が大きいときには記憶装置の無駄が大きい．たとえば，容量 1K，9K，33K，および 121K のジョブが多数実行されるときには，次のように区画が割付けられる：

区画	区画容量	ジョブ容量	無駄な領域
1	8K	1K	7K
2	32K	9K	23K
3	32K	9K	23K
4	120K	33K	87K
5	520K	121K	399K
	712K	173K	539K

この場合，すべての区画が最適状態で割付けられてはいるが，使用可能の 712K の内でわずか 133K だけが実際に使用されているにすぎない．かくして，記憶領域の 75% 以上が無駄になっている．もちろんこれは極端な例であるが，実際のシステムにおいても同様な状況が発生し得るのである．

3.3.2.2 動的な分割

動的とは，ジョブ処理中にジョブの容量に区画の容量を合せて，区画が生成

区画番号(保護キー)	容量	番地	状態
1	8K	312K	使用中
2	32K	320K	使用中
3	32K	352K	未使用
4	120K	384K	未使用
5	520K	504K	使用中

図 3.6 静的区割指定表

されることを意味する．動的割付け方式は，オペレーティング・システム自体が関係しないような場合によく使用される．この技法は，**記憶領域分割**とか，**動的記憶域割付け**，**被制御記憶域**，あるいは**くず拾い**などと呼ばれてきた．(たとえば，PL/I の ALLOCATE 文が実行されたときには，新しい記憶域が割付けられなければならない．逆に，FREE 文が実行されると記憶域は解放される．この種の処理は区画の中で行なわれ，オペレーティング・システムが関係する必要はない．)

動的な分割割付けを遂行するには多数の方式がある．空いている領域と割付けられている領域のそれぞれについて，特に，容量，番地，および参照の制限を登録している表を作成しなければならない．

図 3.7 はこの目的のための表の構成を示している．2つの表——1つは割付けられた領域のためのもので，他は割付けられていない未使用の領域の状態を納める——が使用されている．表は必ずしも完全に記入されているとは限らない．状態部は対応している欄が現在使用中か（割付けられて使用可能の領域の情報を納めている)，あるいは使用されていないか（“空欄”）を示している．

動的な割付けの例を図 3.8 に示す．ある時点では 3 区画が割付けられ，各領域の容量に対応したジョブを収容していた（図 3.8a)．次に，さらに，3つのジョブが多重実行するために選ばれ，割付けられていない空き領域から適当な容量の新しい区画が作成された（図 3.8b)．最後に，ジョブが完了すると対応した区画は解放することができる．図 3.8c はジョブ 2 と 3 が完了した後の記憶装置の状態を示している．

割付けと解放の機能を果たすには各種のアルゴリズムが存在する．図 3.7 のような割付けを行なうアルゴリズムの例を図 3.9 および図 3.10 に示す．割付けに際しては幾つかの手順を経なければならない．第1，要求された区画以上

区画番号（保護キー）	容量	番地	状態
1	8K	312K	割付けられている
2	32K	320K	割付けられている
3	—	—	空欄
4	120K	384K	割付けられている
5	—	—	空欄
	---	---	

割付けられた区画の状態表

空き領域	容量	番地	状態
1	32K	352K	空いている
2	520K	504K	空いている
3	—	—	空欄
4	—	—	空欄
	---	---	---

空き領域状態表

図 3.7 動的区割指定表

の容量を持つ空き領域を探さなければならない．第2，その領域が必要とするより大きければ2つに分割し，その1つを割付けねばならない．もう一方は元より小さい空き領域となる．その反対に区画が解放されたときは，隣り合った空き領域を，小さな複数の領域のままで置かずに，1つの連続した大きな領域に合併する（たとえば，図 3.8bとcのジョブ3の区画を見よ）．

図 3.7に示した表は記憶装置の使用状況を把握する1つの方法である．実際の組込みでは別の手法が採用されることもしばしばある．問題は，区画表と空き領域表の欄の個数が任意の時点の区画数と空き領域数に依存するということであり，これらの数は一定ではないということである．単純な表を用いる手法は，多くの欄が通常使用されていないので，多少効率が悪い．もっと優れた方式は，表の情報を各区画や空き領域に付加することである．そして，この欄を番地ポインタを用いて結合する．この手法は第7章で示した例題のオペレーティング・システムに採用されている．（読者は第 7.8 節を見ておかれるがよい．）

図 3.9 および 図 3.10 に示したアルゴリズムの変形として一般的なのは，

次節で概説する第1適合と最適適合（Johnson, 1973；Knuth, 1968）の2つである．

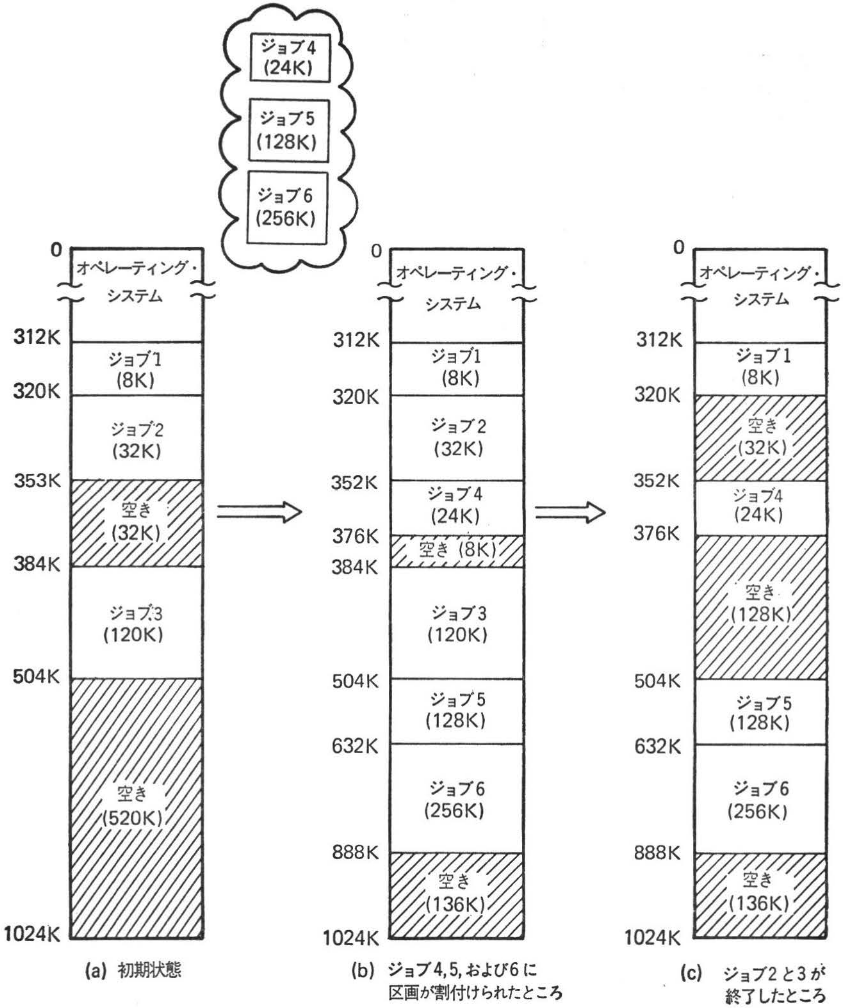


図 3.8 区割の割付けと解放

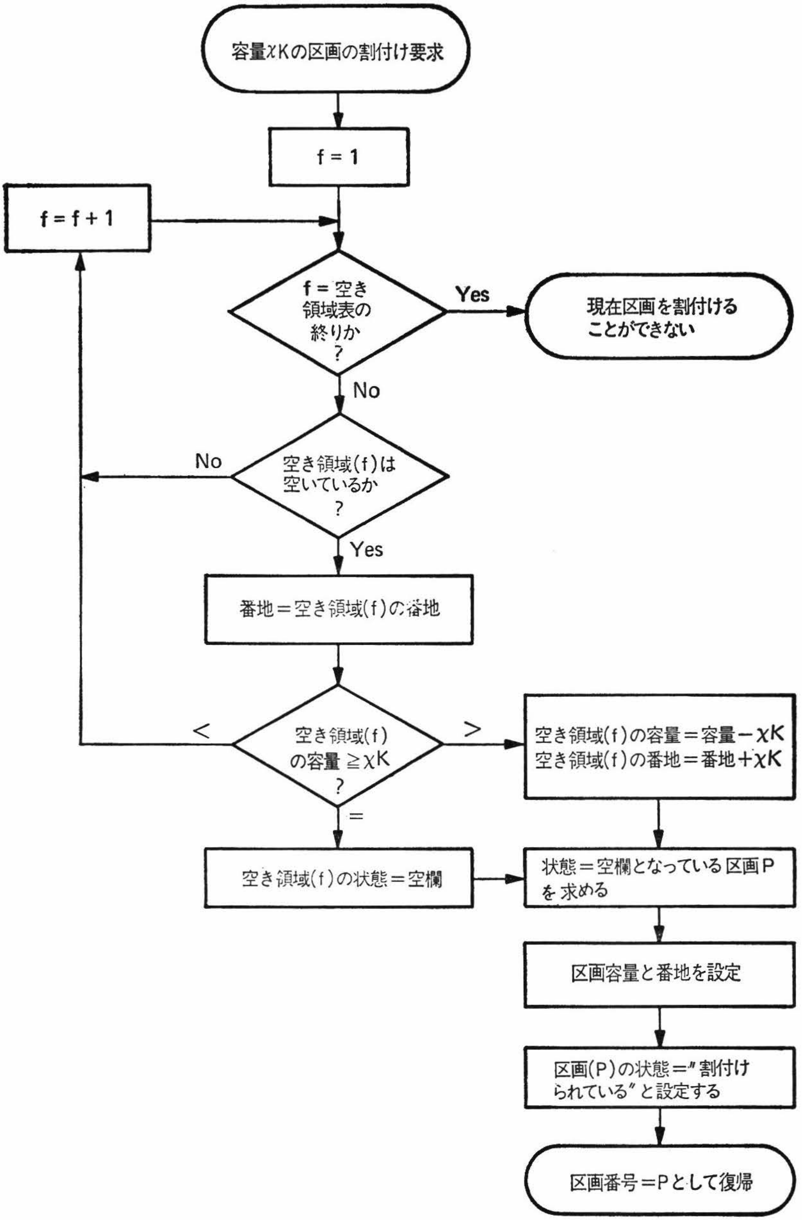


図 3.9 区割割付けのアルゴリズム

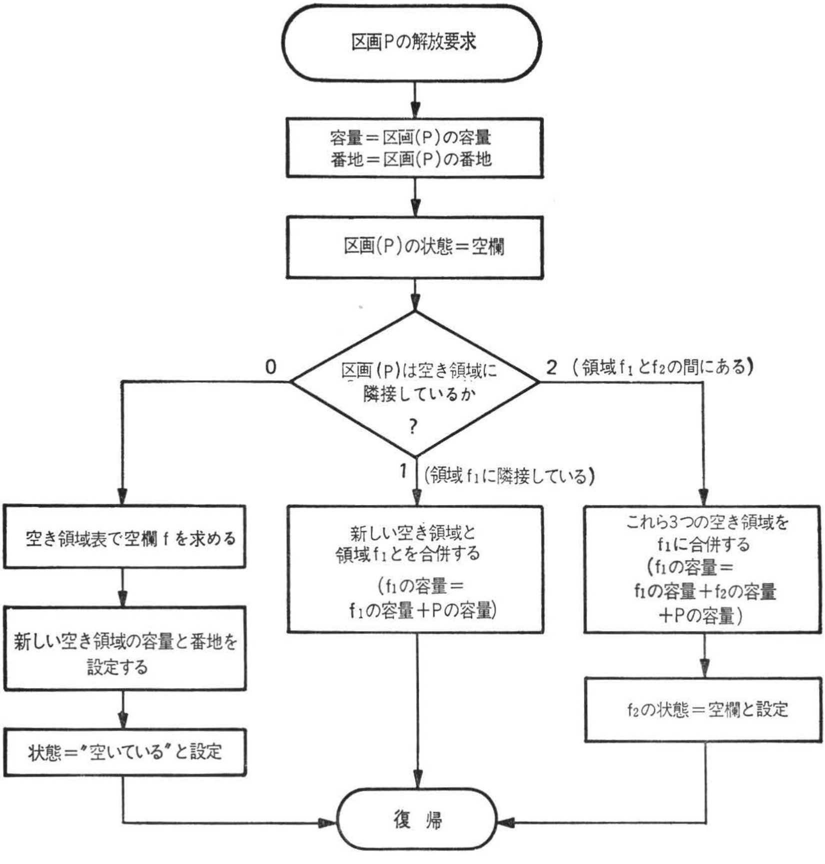


図 3.10 区割解放のアルゴリズム

3.3.2.3 第1適合の割付けアルゴリズム

第1適合のアルゴリズムでは，空き領域表は番地によって分類されている(すなわち，各欄には記憶番地の順に空き領域が対応している)．この順序づけは，上に述べたように表の欄を連鎖しておくときわめて容易に行なうことができる．区画を割付ける必要のあるときには，番地の小さい空き領域から始めて，容量が適合する空き領域が最初に見つかるまで探す．

第1適合の技法には主として2つの利点がある：

1. 解放のアルゴリズム（図 3.10）で，この区画が空き領域に隣接しているか否かは，表の半分だけを探すことにより判明する．（また，記憶番地の小さい方の空き領域が隣接して存在している場合には，それがつねに先に見つかっている．）
2. この技法では番地の小さい方の空き領域を頻繁に使用する．このようにすると，大きな空き領域が番地の大きい方に形成されることになり，非常に大きい容量の区画が必要とされるとき，その容量を満足する空き領域が見つかる確率が大きい．

3.3.2.4 最適適合の割付けアルゴリズム

最適適合のアルゴリズムと第1適合のアルゴリズムで異なっているのは空き領域表が容量で分類されていることだけである（すなわち，第1欄は最小容量の空き領域に対応している）．このようにすると，必要とする区画の容量を満足する空き領域が見つかればこれが**最適適合**となっている（すなわち，この空き領域の容量が必要とされる区画の容量より小でなくて最も値が近いものである）．第 7.8 節で述べている技法は最適適合のアルゴリズムである．

最適適合の技法の主たる利点は3つある：

1. 平均的には，最適適合する空き領域は表の半分だけを探せば見つかる．
2. 要求した容量に等しい空き領域が存在するときには，この領域が選択される．第1適合の場合には，必ずしもこの領域が選択されるとは限らなかった．
3. 要求した容量に等しい空き領域が存在しないとき，区画は可能な限り小さな空き領域から作成され，後で起こるかもしれない容量の大きい区画の要求を満足するために必要な容量の大きい空き領域を破壊せずにすむ．

空き領域は通常，要求の容量に等しくないから，2つの部分に分割しなければならない点に最適適合法の主たる欠点がある．最適の条件を満たそうとすることによって，分割後の空き領域はしばしばきわめて小さくなり，他の役に立たない．いいかえると，1つの大きな空き領域ではなく，非常に小さな領域が無数に生成される傾向がある．

3.3.2.5 断片化の問題

分割割付けアルゴリズムを採用するときは幾つかの要素を考慮しておかねば

ならない．速度と単純性もその1つである．しかし，これらは容易に判定できる．より重要なことは断片化の効果である．**断片化**とは分離した空き領域が多数発生することをいう（すなわち，空き領域が小さな断片となることである）．たとえば，図 3.8c を参照すると，容量 138K の区画が要求されたとき，これに区画を割付けることができない．空き領域は合計で 296K バイト存在するが1つで 136K 以上の空き領域が存在しないのである．

断片化は分割割付け以外の技法を用いて克服することができる．これらの技法について，第 3.4 節および第 3.5 節において再配置とページングの2つの方法を述べる．

分割割付けにおける断片化の問題は，使用するアルゴリズムを注意して選択することによって最小とすることができる．これまでに論じてきた静的な割付け，動的な第1適合割付け，および動的な最適適合割付けのそれぞれについて，その割付けが最適であるような，あるいは不適であるような状況が存在する．この“状況”とは一連の分割割付けと解放の並び方のことをいう．

一例として；

ジョブ1が 140K を要求
ジョブ2が 16K を要求
ジョブ3が 80K を要求
ジョブ1が終了
ジョブ3が終了
ジョブ4が 80K を要求
ジョブ5が 128K を要求
など……

あるアルゴリズムが要求を即座に満たすことができず（すなわち，他の区画が解放されるまでは，区画を割付けることができない），別のアルゴリズムではその要求を満たすことができるようなとき，そのアルゴリズムにとってこの状態は“悪い”と考えられる．空き領域が 256K バイト与えられているとして，読者は自分で上に示した一連の要求を第1適合および最適適合によって割付けてみよ．この要求は両方のアルゴリズムにとって悪い状況であるか，その理由は何か．各アルゴリズムにとってよい状況と悪い状況を作り出してみよ．静的な区画容量を適当に選んで，静的割付けについても考察してみよ．

“最適な”アルゴリズムの選択は一連の要求の性質によって影響される（たとえば，すべてのジョブの容量が等しいとか，たいていのジョブは 64K であ

り，2～3のジョブが 256K であるなど). 第1適合や最適適合のほかに多くのアルゴリズムが存在する (Knuth, 1968；Madnick, 1966；Ross, 1967).

3.3.2.6 多重分割割付けアルゴリズム

多重分割割付けアルゴリズムを用いると断片化の問題を軽減することができる．通常1つのジョブは幾つかの別々の記憶域の区画から成っている．すなわち，別々のサブルーチンやデータ配列など．個々の区画内では論理的に連続でなくてはならない (たとえば，$A(i+1)$ は $A(i)$ に続いている) が，適当な保護機構があれば，区画の間は連続していなくてもよい．このようにすると，100K の記憶域を必要とするジョブは実際には5つの20K の区画から構成されることもある．このジョブは，1つの100K の区画，あるいは5つの 20K 区画，あるいは2つの 40K の区画と1つの 20K 区画，…，に割付けることができる．この手法は，1つ以上の区画をジョブに割付けることが可能であることから**多重区画割付け**と呼ばれる．さらに，この種の方式を用いると，ジョブが使用している区画と新しい区画とが連続である必要がないので，たとえば特殊な誤り処理サブルーチンをロードしたりするために，ジョブの実行中に追加の記憶域を要求することが可能となる．前述と同様の記憶割付け方式は薄命の IBM OS/360 VMS (可変記憶域システム) で採用された．(このシステムは公式に発表されたり，顧客で使用されたことがなく，単一割付け方式の OS/MVT で置換えられた.)

多重区画割付けは記憶管理により大きい融通性を与えたが (すなわち，ジョブの要求を満たすのに1つ以上の方法が存在する)，断片化の問題は解消しなかった．平均的には，空き領域の各区画は一層小さく，より多くの領域が生成される．これらのあるものは実際には役に立たない．

ジョブのための最小の区画容量の範囲が大きいとき (たとえば4バイトの要素が 50,000 個必要な配列は 200,000 バイトの連続領域を必要とする)，これらの大きい領域のための空き領域が見つかる確率が小さい．たとえば，100K バイトを要する1つのジョブを，100万バイトの記憶装置を切りきざんで作った 2K の部分を 50 個使用して割付けることは可能であるが，20K バイト以上の空き領域は存在しないであろう．20Kバイトというのは他の多くのジョブで必要な最小容量よりも小である．この例は極端な例であるが，このような状況は実際にも見られている．

すべての部分の容量がほぼ等しい場合には，多重区画割付けはより一層魅力的である．この手法の別の実現方法であるセグメンテーションについては第

3.7 節にて述べる.

3.3.3 利　　点

分割割付けには主として3つの利点がある：

1. 多重プログラミングを可能にする．したがって，処理装置と入出力装置の利用率がよくなる.
2. 特殊で高価なハードウェアを必要としない.
3. 使用するアルゴリズムは簡単で組込みが容易である.

分割システムでシステムが使用する記憶容量はどれ位であろうか．これは，少なくとも2つの要素が関係する：

1. ジョブの平均的な容量
2. 処理装置の利用率の希望値

幾つかの IBM 360/65 および 370/55 の計算システムで観測されたことは (Lehman, 1968)，典型的なジョブで約 128K バイト必要であり，単独で実行したとき，入出力待ちで65%の時間を費やしている．第 3.2.2 節の効率のモデルと表 3.1 からは，入出力待ち時間を 10% 以下にするためには，4つのジョブを多重プログラムして 8% 以下，5つのジョブで 3% 以下であることが示されている．このことから，記憶容量は 640K バイトが適当である．しかし，断片化を補償するために保留分が必要である．⅓が断片化のために無効になっていると仮定すると，必要量は 960K バイトになる．したがって必要となる記憶容量の見積りは約 100 万バイトということになる．もちろん，記憶装置はオペレーティング・システムを格納するためにもっと大きくなければならない．典型的な値として，複雑な多重プログラミング・システムのためには 256K から 512K の範囲が必要である．しかし，オペレーティング・システムによってはこれ以下のもあればこれ以上のもある.

3.3.4 欠　　点

分割割付けには幾つかの欠点があり，記憶装置の利用率を低下させている：

1. 断片化は重要な問題である．記憶装置の利用率が10%以下となるようなジョブの並び方が存在する．断片の程度はジョブの並び方と使用するアルゴリズムに関係する.
2. 記憶領域が断片とならないまでも，1つの空き領域が小さすぎて区画

として割付けられないことがある．たとえば，512Kの記憶領域があり，大半のジョブは256Kであるが，一部は300Kであるような一連のジョブがあったとする．300Kの区画が割付けられると，残りの212Kバイトは他のジョブにとって小さすぎるので，無駄となる．

3. 2つ以上のジョブを格納するためには，単一連続割付けシステムよりも大きい記憶装置が必要である（より複雑なオペレーティング・システムの場合も同様である）．

3.4 再配置型分割割付けの記憶管理

図 3.8cで示したような断片化の問題を解決するには，定期的にすべての空き領域を1つの連続した領域に結合すればよいことは明らかであろう．これは図 3.11 に示したように，割付けられている区画をすべて移動して連続させればよい．この処理は**詰め合せ**と呼ばれる．

この処理は概念的には単純であるが，ジョブの区画を移動させるとそのジョブが新しい場所で依然として正しく動作することが保証できない．これは次のような場所に関係するものが多数存在するからである：

1. ベース・レジスタ
2. 記憶場所を参照する命令
3. 引数表
4. 番地ポインタを使用するデータ構造（リスト，チェインなど）

正しく動作させるには，場所に関係するものをすべて適当に修飾しなければならない．たとえば，図 3.11 でジョブ4は352K番地から320K番地に移された．ジョブ4に含まれる番地はすべて32Kだけ減じなければならない．この格納場所に関係する番地の調整を**再配置**と呼ぶ．これは再配置型ローダの動作と類似している（Donovan, 1972；Presser, 1972）．

困ったことに，ジョブの区画の再配置はきわめてむずかしいことである．特に，変更しなければならない番地ポインタをオペレーティング・システムはどのようにして区別すればよいのだろうか．ジョブ4の区画に数364,000があったとしても，これは再配置すべき番地ポインタなのであろうか．あるいは会社の支払金額を示す変数なのであろうか．このためのソフトウェアの技法と規約を設けることは可能であるが（Reiter, 1967），これらは通常能率も悪く，制限も多い．

再配置の問題を解決するのに，再配置すべきジョブを全部ロードし直して，

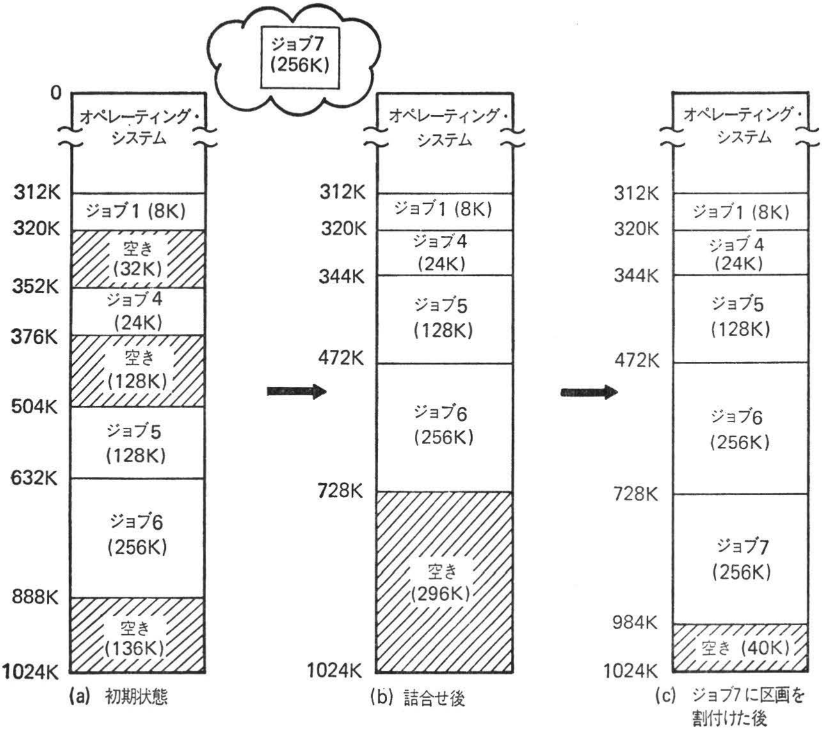


図 3.11 再配置による区割の詰め合せ

最初から実行することが考えられる．最初に“静的に”再配置するには再配置型ローダを用いればよい．しかし，あまりにも原始的であるうえに（すなわち長時間の計算をやり直さねばならない），プログラムがすでに非可逆的な処理を実行していれば，このように再実行することは適当でない（たとえば，読者の名前がアルファベットの始めの方にあれば，2枚以上の給料支払小切手を受取るかもしれない）．

この再配置の問題を取扱うためのハードウェアの技法が幾つか開発されている．これらの多くは写像された記憶域の考えによるものである．すなわち，ジョブから“見える”番地空間は必ずしも実際の物理的な記憶番地とは等しくない．この概念は時として**仮想記憶**と呼ばれるが，ここではこの言葉を第3.6節

で紹介する特定の番地写像を意味するものとして残しておく．第**3.4.1**節では使用可能の技法について述べる．

詰め合せと再配の処置理以外は，再配置型分割割付け方式は前述の分割割付け技法にきわめてよく似ている．

3.4.1 ハードウェア

再配置の問題を解決するには普通2つの方法があり，両者共特別なハードウェアを必要とする．1つの手法はデータの型の概念を用いるもので，すべての記憶番地に格納されているデータの型を記録しておくものである．たとえば，すべての語に2ビットを追加すると値の型を示すことができる（たとえば，00＝整数，01＝浮動小数点数，10＝文字，11＝番地ポインタ）．これらのビットは普通の利用者からはわからず，ハードウェアによって自動的に設定される．たとえば，文A＝BはAにBの値を設定するだけではなく情報の型をも複写する．このようにすると，番地ポインタの処理と移動が可能となる．詰め合せと再配置を行なう必要があるときには，オペレーティング・システムは特権命令を実行して型の符号を検査し，変更を必要とする番地ポインタを見つけることができる．この型のハードウェアは Burroughs 5500 および 6700 のような計算機に備わっている．

データの型によって番地の写像を行なうことは，利用者プログラムにおいて番地ポインタが特別に取扱われているために可能である．それらは任意に検査したり処理したりすることはできない．したがって，プログラムは格納場所を決定することができないので（たとえば，このプログラムはX番地にロードされるかといったことはわからない），利用者は番地に関しては何もすることができない．

データの型を使用する手法には幾つかの欠点がある．全部の語に余分のビットが必要であり，全部の語についている型のビットを検査しなければならないので詰め合せが遅い．また，現代の計算機（特に，IBM 370）の一般的な構造との互換性がない．

別の解決法は，ベース限界再配置レジスタを用いて動的な再配置を行なう方法である．この技法は DEC PDP-10 や，Univac 1108，Honeywell 6000 シリーズなどの幾つかの計算機で採用されている．**ベース再配置レジスタ**と**限界レジスタ**の2つの特殊な特権レジスタがあり，オペレーティング・システムだけが参照可能である．記憶装置が参照される度に，ベース再配置レジスタの内容が自動的に実効番地に加算される（実効番地とは処理装置で計算された最終

的な参照番地である．この計算にはインデックスや IBM 370 型のベース・レジスタによる修飾が含まれる）．

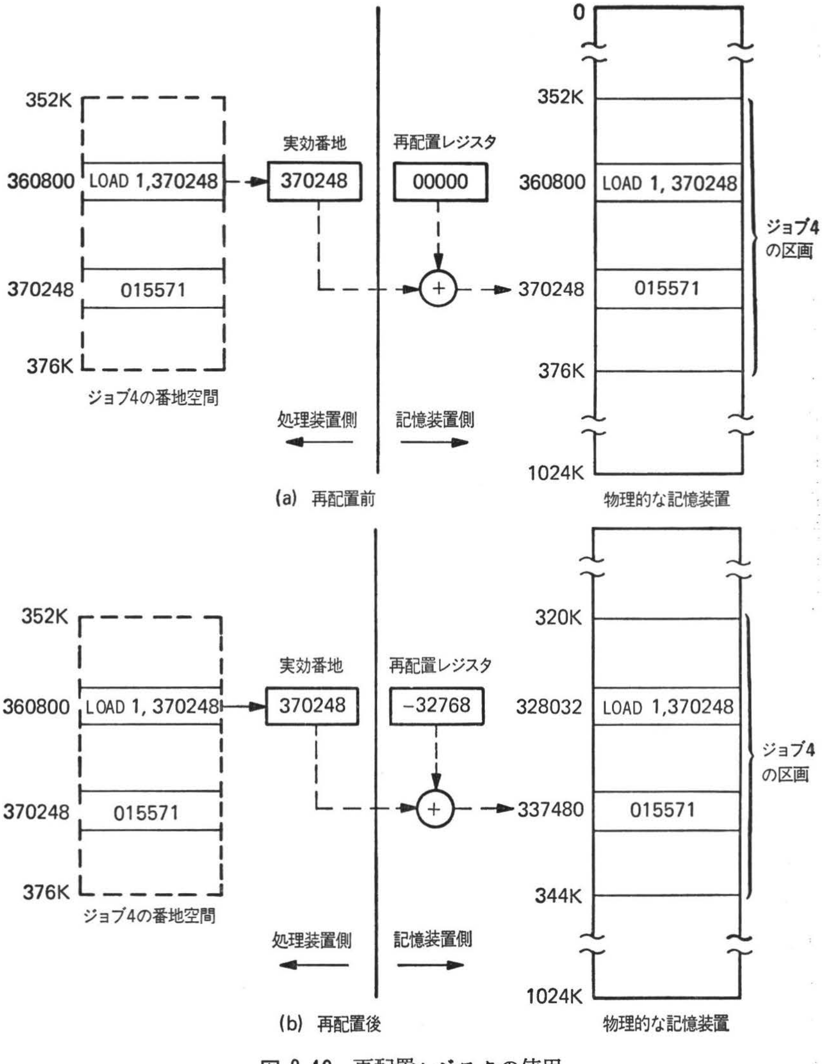


図 3.12 再配置レジスタの使用

図 3.12aの左側には図 3.11aのジョブ4の番地空間が示されており，右に示したジョブの区画の物理番地とは区別されている．再配置を行なう前には，360800 番地の命令 LOAD 1,370248（番地 370248 からレジスタ1にロードする）はレジスタ1に値 015571 をロードする（読みやすくするために，IBM 370 で行なっているようなベース＋変位の記法を用いずに，命令には直接実効番地を記入しておく）.

再配置の後では，図 3.12bの LOAD 命令は 328032番地に置かれている．命令やレジスタには変更がないので，目的とするデータの値は 337480 番地に移動しているにもかかわらず，ロードすべきデータの実効番地は依然として 370248 である．正しい結果を得るために，オペレーティング・システムは再配置レジスタに −32768（320K−352K=−32K）を設定しなければならない．命令 LOAD 1,370248 が実行されるときには，−32768 が自動的に実効番地 370248 に加算されて，実際に参照されるべき物理番地が決定される．この場合には，370248−32768=337480 となる．このようにして，目的の値 015571 がレジスタ1にロードされる．注意すべきことは，ジョブの番地空間の中で変更されたものは何もなく，また，プログラムは実際の物理番地を知らないことである．LOAD, ADD, BRANCH などの命令は全部，実際には 320K に移動されているにもかかわらず，区画が 352K にあるかのごとく実行される．

この再配置は，各命令が実行されるときに自動的に行なわれるので，**動的再配置**と呼ばれる．

保護は，IBM 370 のロックとキーのような技法を用いても実現できるが，普通にはジョブの番地空間の最大番地を（前の例では 376K）限界レジスタに設定する方法が用いられる．ジョブが限界レジスタに指定された番地を越えて参照しようとすると，保護割込みが発生する．

3.4.2 ソフトウェアのアルゴリズム

ジョブの番地空間の開始番地は区画の物理番地と独立であるから，番地空間を0番地から開始しておくのが便利である．すなわち，各ジョブは記憶装置の0番地からロードされると“思って”いる．このようにすると，区画の番地がどこでも，ジョブ4の番地は図 3.13のように設定される．この規約を採用すると，再配置レジスタの値はすべて正である．さらに，ハードウェアで実効番地が負となったり限界レジスタの値を越えたことを検出できるので，保護を行なうことができる．

論理的にはベースと限界レジスタが各区画に必要となるが，これは必ずしも

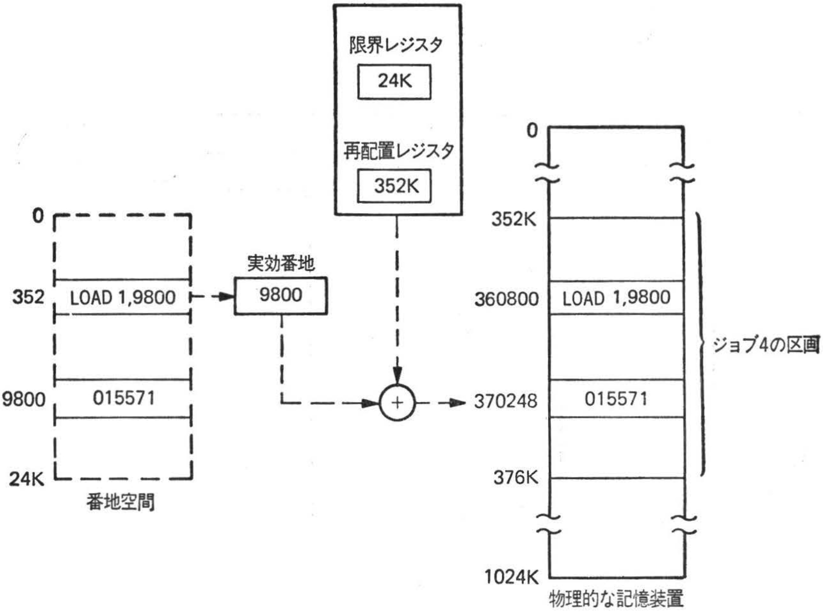


図 3.13 ジョブ4の番地空間が0番地にロードされていると考えたとき

必要でなく，また経済的なやり方でもない．ジョブの実行を開始するに先立って，1組のレジスタにジョブの区画に対するベースと限界とを設定する．多重プログラミングを行なっていてジョブがある事象を待合せなければならないときには，ジョブを再開するときに再配置することができるように，これらのレジスタの内容が（PSW などの状態情報と共に）保存される．

分割割付けアルゴリズムは，基本的には第3.3節で論じたのと同じことが問題となる．1つの大きな問題は“いつ詰め合せを実行したらよいか”である．これには2つの考え方がある．第1は，区画が解放された直後に必ず詰め合せを行ない，つねに1つの連続した空き領域だけを残し，断片のないようにする方法である．その結果空き領域表の管理と区画の割付けはきわめて容易となる．しかし，詰め合せは時間のかかる処理である．たとえば，毎秒1,000,000バイトの割合で区画を移動させることができたとしても（IBM 370 の MVC や MVCL 命令を使用する），1024 K バイトの記憶装置の中央で解放された小区画を詰め合せるためには約0.5秒の時間がかかる．

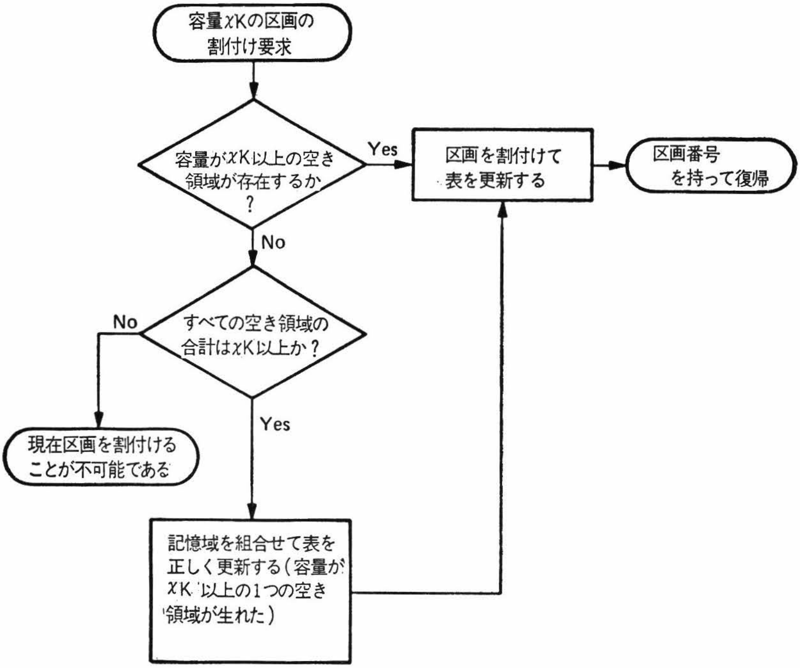


図 3.14 再配置型区画割付けの概略

第2の方法は，第3.3節で述べた再配置を行なわない分割割付けアルゴリズムと全く同様に動作し，必要な空き領域がなくなったときに詰め合せを行なうものである．詰め合せの回数は上述の即時詰め合せ方式よりもはるかに少ないが，表の管理ははるかに複雑である．図 3.14 はこのアルゴリズムの概略を示している．

3.4.3 利 点

再配置型分割割付け方式は断片を除去し，より多くの区画の割付けを可能とする．この方式では多重プログラミングの多重度を大きくすることができ，その結果記憶装置と処理装置の利用率が増大する．

3.4.4 欠 点

幾つかの欠点を考慮しておかねばならない：

1. 再配置のハードウェアは計算機の経費を増大させ，速度を低下させる（通常はほんのわずかである）.
2. 詰め合せ時間が相当大きい.
3. 詰め合せを行なってもなおかつ空き領域が要求された区画の容量よりも小さいことがあるので，記憶域のある部分は依然として使用されることがない.
4. 分割割付けの場合と同様に，記憶装置には参照されることのない情報までも格納されることがある．さらに，ジョブの区画の容量は物理的な記憶容量で制限される.

3.5 ページングによる記憶管理

再配置型分割割付け法は，番地写像により別個の空き領域を1つの連続した空き領域に統合することによって，断片化の問題に対する1つの解決法を与えた．このようにしなければならない理由は，区画は連続した領域でなければならないからであった．多重区画割付けのアルゴリズムによってこの連続性の要求を緩和することができたが，いつも断片を減少できたわけではなかった．連続性の要求を取除く他の方法はページングによって記憶管理を行なうことである.

ページングを行なう記憶管理法では，各ジョブの番地空間は**ページ**と呼ばれる等長の部分に分割され，同様に，物理的な記憶領域も**ブロック**と呼ばれる等容量の部分に分割される．そして，適当なハードウェアの写像機構を設けておくことによって，任意のページを任意のブロックに置くことができる．ページは論理的には（すなわち，利用者プログラムにとっては）連続したままであるがそれに対応するブロックは必ずしも連続している必要はない．再配置型の分割割付け方式と同様に，利用者プログラムの番地空間には番地写像が直接影響しないので，利用者プログラムには何の影響もない.

番地空間から物理的な記憶域への写像を行なうハードウェアには，各ページごとに別個のレジスタが必要である．これらのレジスタはしばしば**ページ・マップ**とか**ページ・マップ表**とか呼ばれる．これらは特別のハードウェアのレジスタであってもよいし，記憶装置の特定の部分であってもよい.

各ページは別々に再配置することが可能であるので，ジョブの区画が記憶装置の中で完全に連続している必要はない．1つのページの中の番地だけが連続していればよい．明らかに，ページの大きさをどのように選ぶかがこの方式の

有効性に本質的な影響を与えることになる．ページが大きすぎるとジョブの区画の容量に近くなり，この方式は再配置型割付けにほかならなくなる．ページが小さすぎると多数のページ・マップ・レジスタが必要となり，計算機システムの経費を大幅に増大させる．これらおよびその他の点から，多くのページング・システムでは1Kから4Kバイトのページを使用している．

1000バイトのページを用いた簡単な例を **図 3.15** に示す．3000バイトの番地空間を有するジョブ2は3ページに分割されている．ジョブ2に関するページ表はページの番地を示している．この場合には，ページ0はブロック2に，ページ1はブロック4に，ページ2はブロック7に入っている．ジョブ2の番地空間の0518番地（ページ0の518バイト目）にある LOAD 1,2108 という

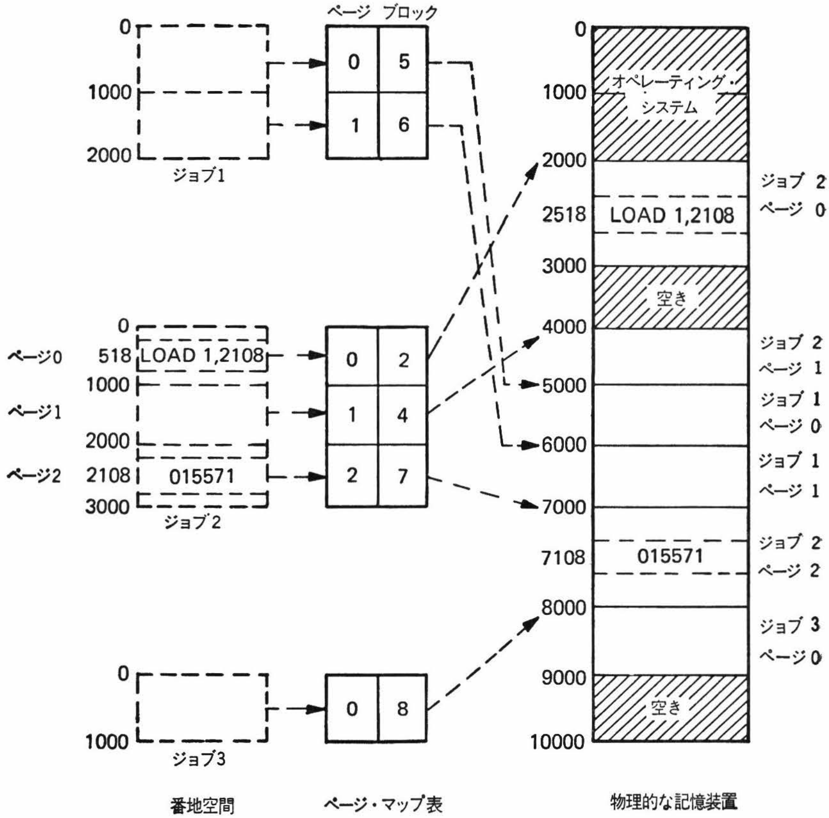


図 3.15　ページ割付けを行なう記憶装置

命令は，実際には物理番地2518（ブロック2の518バイト目）に格納されている．同様に論理的な2108番地のデータ015571は，物理番地7108に格納されている．

ページングによる記憶管理法は，区画を物理的に移動させずに断片化の問題を解決する．たとえば，図3.15では2000バイトの記憶域が使用できるが，それらは連続していない．2000バイトを必要とするジョブ4があったとすると，再配置型割付けで行なったようにして，記憶域を詰め合せて1つの2000バイトの空き領域を作成することもできる．別の方法としては，ジョブ4の2ページを，ページ0はブロック3に，ページ1はブロック9にというように，空いているブロックに割付けることもできる．ページ表を適当に設定しておけば，区画を物理的に移動しなくても，依然として番地空間は連続しているように見える．この種の方式は，たとえば，XDS 940，XDS Sigma 7，CDC 3300などの，現代の幾つかの計算機で採用されている．

記憶管理の4つの機能は次のようにして実行される：

1. 状況を把握すること——2組の表を用いて行なわれる．
   a．ページ・マップ表——番地空間ごとに1つの表が設けられ，1ページごとに1つの欄がある．
   b．記憶ブロック表——システムに1つの表が設けられ，記憶ブロックごとにそのブロックの使用状況（たとえば，割付けられているとか使用可であるとか）に関する情報を記入する欄がある．
2. どのプロセスに記憶域を割付けるかを決定すること——これはおおむねジョブ・スケジューラによって決定される．記憶域は，空いているブロックで単に先に見つかったものを利用者に割付けるだけである．
3. 割付け——ジョブのページはすべて割付けられたブロックにロードしなければならない．また，ページ・マップ表と記憶ブロック表の適当な欄にも記入しなければならない．
4. 解　放——ジョブが完了したとき，ブロックを空き状態に戻し，ブロック表の欄を修正しておかねばならない．

3.5.1 ハードウェア

各命令の実効番地を適当な物理番地に写像するハードウェアの機構が必要である．この機能は再配置レジスタのハードウェアと類似している．“レジスタ”は各ページごとに必要である．これらは高速のレジスタでもよいし，物理的な

記憶域の一部でもよい.

3.5.1.1 高速のページ・マップ・レジスタ

100万バイトの記憶装置があり，ブロック容量が4000バイトであれば，最大250ページを写像する必要がある．しかし，高速のレジスタを250個使用するのはきわめて高価につく．経費上の理由でこの技法にほとんど採用されることがない．それにもかかわらず，幾つかの場合にはこの高速レジスタ方式は経費的な問題を越えてもっと大きな魅力が生じる．第1に，各ジョブの番地空間の容量が，たとえば100Kバイトというように，比較的小さく限定されている場合には，各ジョブの番地空間に必要となるレジスタは25個以上必要でない．第2に，一時に1ジョブだけが実行されるので，1組のハードウェアのページ・マップ・レジスタだけが必要である．多重プログラミングによって，処理装置が新しいジョブに切換えられたときには，これらのレジスタを退避してから内容を消去しておかなければならない．この技法は XDS 740 で採用されている.

3.5.1.2 ページ・マップ表

大容量の番地空間に対して，ページ・マップ・レジスタをたえず設定し直すオーバヘッドを除去するために，記憶装置の中にページ・マップ表を作ることが一般に行なわれている．次に述べるハードウェアは IBM 370 で使用されている動的番地変換(DAT)——ページ写像の別名——を簡単にしたものである.

番地写像を単純にするために，ページの大きさは，通常，たとえば，1024 (1K) バイト，2048 (2K) バイト，4096 (4K) バイトといった2の累乗の値が選ばれる．このようにすると，ページの大きさが4Kバイトであれば，ページ番号は実効番地のビット8〜19で与えられる（図 3.16参照)．実効番地は処理装置で形成され，24ビットの長さである．DAT機構は実効番地を自動的に2つの部分に分ける．ビット8から19は12ビットのページ番号となり，ビット20から31は12ビットのページ内オフセットとなる．ページ・マップ表を使用して，ページ番号はブロック番号に置き換えられ，物理的な記憶番地が生成される.

ページ・マップ表が主記憶に格納されるとすると，ハードウェアはその番地を知る必要がある．このために，“旧”および“新”の PSW や，CAW, CSW などで行なったように特定の番地を割付けておくことも可能である．しかし，このようにすると，処理装置がジョブを切換える度に表全体を変更しなければならない．そのようにする代わりに，ページ・マップ表を記憶装置の任意の番

地に格納できるようにして，その番地を特別のページ・マップ表番地レジスタ(PMTAR) によって表示する．処理装置を新しいジョブに切換えるときには，

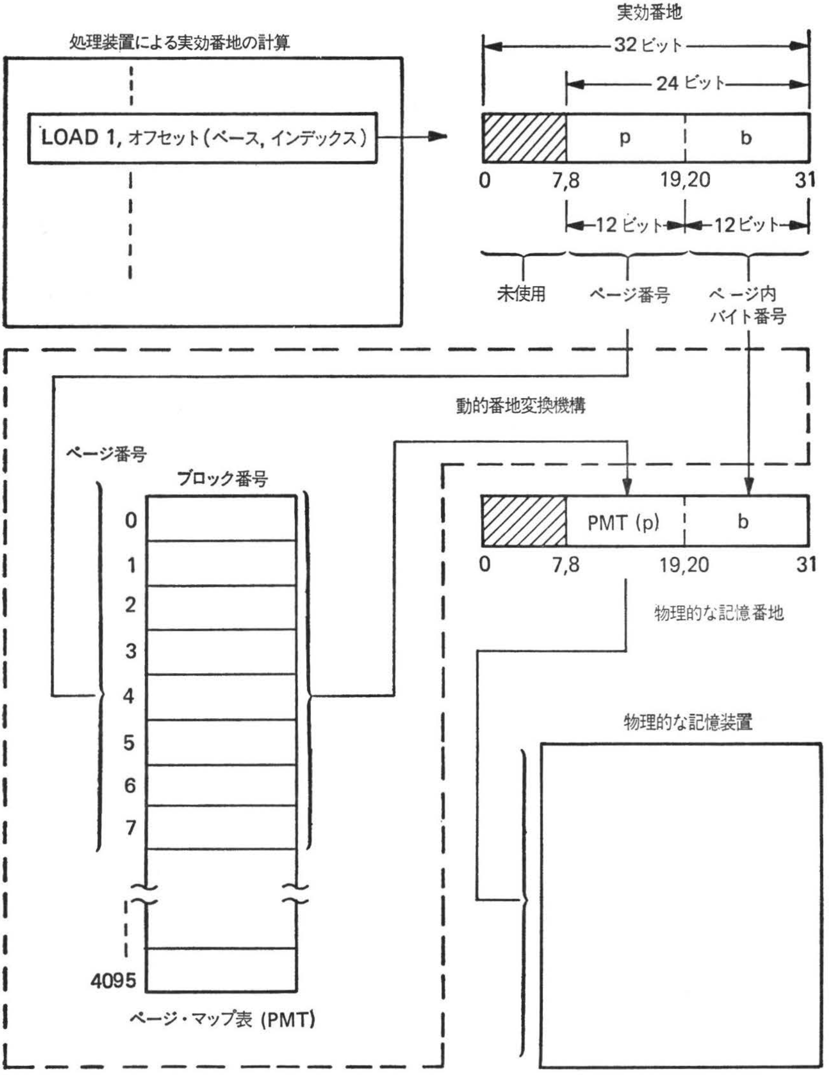


図 3.16 実効番地から物理的な記憶番地を決定する機構

このページ・マップ表番地レジスタが新しいジョブのページ表の番地を指すように変更するだけでよい．

図 3.15 に示した簡単な例の一部を図 3.17 に書き直して，ページとブロック，ページ・マップ表番地レジスタ，ページ・マップ表の関係を示しておく．

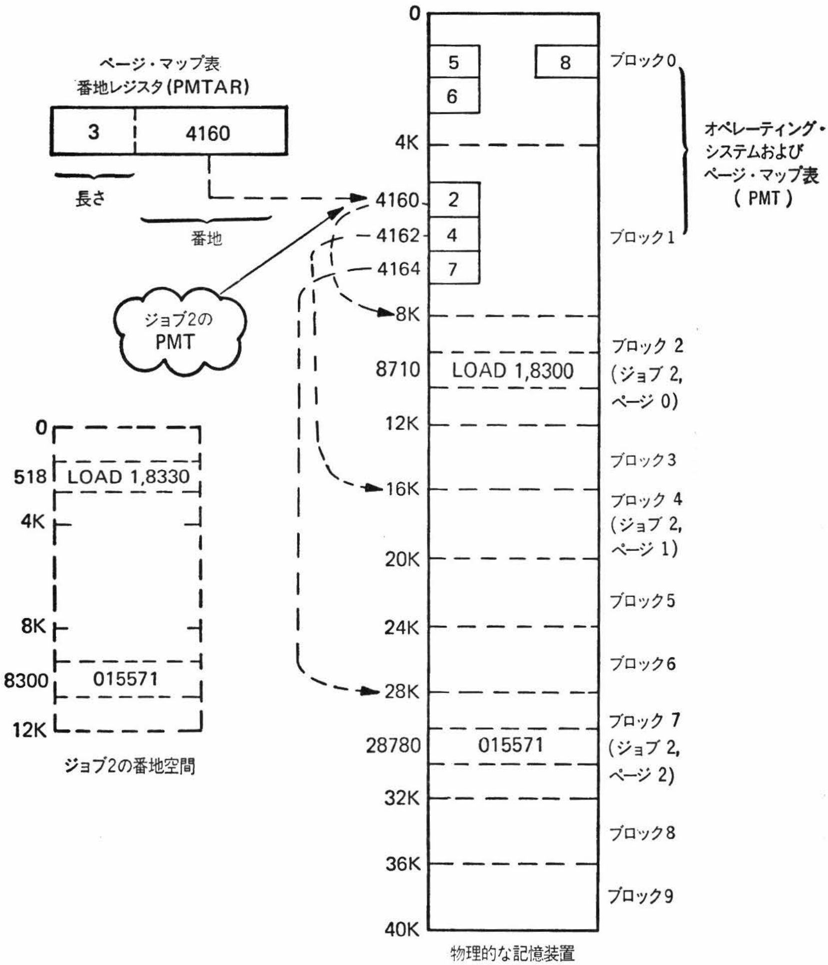
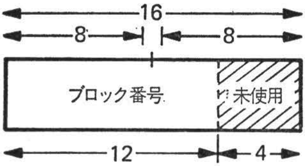


図 3.17　ページ，ブロック，PMTAR および PMT の関係

この図で使用されているページの大きさは 4096（4K）バイトである．ページ表の欄の符号化には種々の方法が存在する．IBM 370 では各欄に格納されるブロック番号は 12 ビットなければならない．この 12 ビットの数は 16 ビットの半語（2バイト）に次のように格納される：



1つのジョブの番地空間の大きさは，IBM 370 では最大 1,600 万バイト(4,096 個の 4K ページ）である．各ジョブのページ・マップ表が 4,096 欄の長さであり，各欄が2バイトとすれば，表の領域として 8,192 バイト必要である．たいていのジョブは 256K バイト（64 ページ）以下であることから，64 欄の分のただの 128 バイトを格納するために 8,192 バイトのページ・マップ表を使用したくない．

この問題は，表を必要なだけの長さに限定することによって解決される．表の長さはページ・マップ表番地レジスタの一部として図 3.17 に示したように格納される．この長さは，ジョブの番地空間の最大長を示しており，第 3.4.1 節で論じた限界レジスタのような働きをする．

3.5.1.3 ページ・マップ表とページ・マップ・レジスタの併用

前節で述べたように，主記憶中のページ・マップ表を使用する動的番地変換のハードウェアを用いると，計算機は通常の半分の速度で実行することとなる（すなわち，100% のオーバヘッドを招く）．記憶装置に対する参照はことごとく番地変換しなければならない．ページ・マップ表が主記憶に格納されているときには，記憶装置の参照は2回のアクセスを必要とする．1回はページ番号をブロック番号に写像し，もう1回は要求するデータや命令の参照を行なう．

高速の写像レジスタとページ・マップ表を併用する方式は，この速度の問題を克服するためにしばしば採用される．少数の（たとえば16個）特殊な高速度のレジスタがページ・マップ表の一部を収容するために使用される．これらのレジスタは，主記憶中のページ・マップ表をアクセスせずにすませるよう使用される．ページ・マップ表の欄の内容が，これらのレジスタへハードウェアで自動的にロードされる．これらの特殊なレジスタとそれに関係するハードウェ

アは，しばしば**連想記憶装置**とか**表の探索バッファ**とか呼ばれる．この機構を用いると写像のオーバヘッドを100%から通常10%以下に減少させることができる．連想記憶はハードウェアで自動的に使用されるので，その存在は，通常，オペレーティング・システムにも利用者ジョブにも見ることはできない．オペレーティング・システムはページ・マップ表だけに気がついている．このような2層の表の利用の基となる幾つかの原理を第 3.6 節および第3.10.3節で論じる．

3.5.2 ソフトウェア

オペレーティング・システムで管理する必要のある表は基本的に3つ存在する．これらは次のとおりである：

ジョブ表（JT）
記憶ブロック表（MBT）
ページ・マップ表（PMT）

各ジョブはジョブ表の中に1つの欄を持ち，これがそのジョブのページ・マップ表の番地と長さ，およびその他の状態情報を表示している．記憶ブロック表は各記憶ブロックの状態――割付けられているかあるいは使用可能か――を表示している．図 3.18 は図 3.15 および図 3.17 に示された状況に対して使用される表を示している．

3.5.2.1 番地空間の割付け

番地空間割付けの簡単なアルゴリズムを図 3.19に示す．番地空間の解放には同じアルゴリズムがほぼ逆向けに使用される．説明を要するステップはN欄のページ・マップ表を割付けることだけである．このような表をどこに作ったらよいであろうか．1つの方法は，ページ・マップ表のために，オペレーティング・システムの中に幾らかの領域を確保しておくことである．この領域を使用して，個々のページ・マップ表が1つの区画であるとして，分割割付けの記憶管理と同様な技法によりページ・マップ表を割付けることができる．ジョブが終了したとき，このページ・マップ表の区画は解放される．

3.5.2.2 部分的に重複する番地空間

再配置型分割割付けの記憶管理と同様に，各ジョブの番地空間は他のジョブと独立で分離している．このようにすると，1つのジョブが他のジョブの番地

ジョブ番号	容量	ページ・マップ表の番地	状態
1	8K	3600	割付けられている
2	12K	4160	割付けられている
3	4K	3820	割付けられている
4	-	-	空欄

ジョブ表 (JT)

ページ番号	ブロック番号
0 (3600)	5
1 (3602)	6

ジョブ1のPMT

ページ番号	ブロック番号
0 (4160)	2
1 (4162)	4
2 (4164)	7

ジョブ2のPMT

ページ番号	ブロック番号
0 (3820)	8

ジョブ3のPMT

ページ・マップ表 (PMT)

ブロック番号	状態
0	オペレーティング・システム
1	オペレーティング・システム
2	ジョブ2
3	空き
4	ジョブ2
5	ジョブ1
6	ジョブ1
7	ジョブ2
8	ジョブ3
9	空き

記憶ブロック表 (MBT)

図 3.18 ページングによる記憶割付けのソフトウェアで使用される表

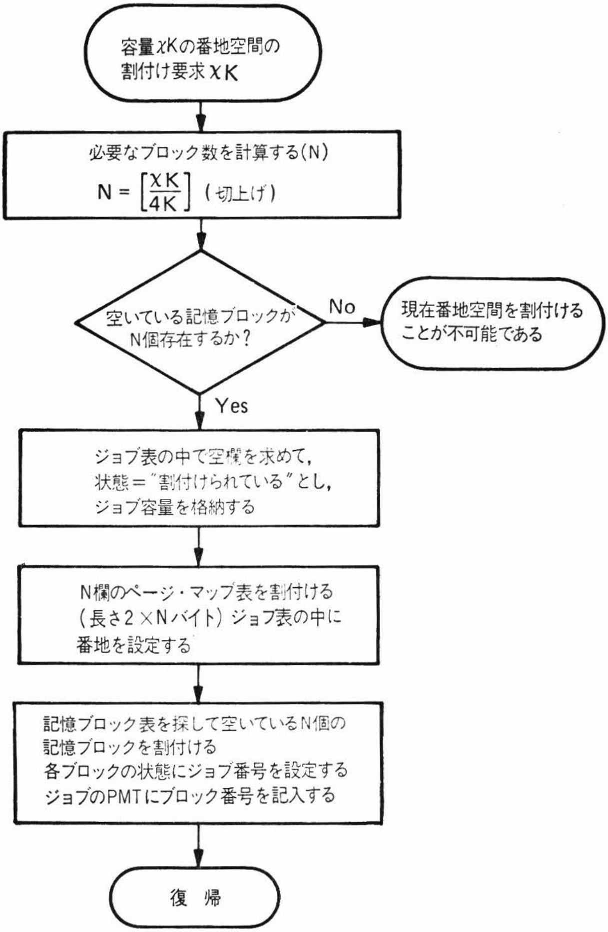


図 3.19 ページングによる記憶割付けのアルゴリズム

空間を納めている記憶域をアクセスすることが不可能であるので，効果的な保護機構を構成することができる．（入出力チャネルは，通常，番地空間の写像のハードウェアを含んでいないので，簡単なロックとキーの保護機構は依然として必要である．）

ある種のサブルーチンやデータ領域をすべてのジョブでアクセスできるようにするために，部分的に番地空間を重複させたいことが時として起こる．たとえば，オペレーティング・システムでは現在の状態を示すデータを格納する領

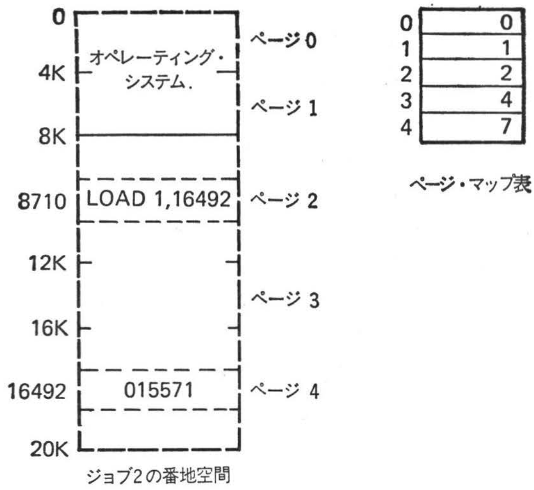


図 3.20 番地空間の重複（図 3.17 参照）

域が作られる．このような重複は図 3.20 に示すようにして行なうことができる．すべてのジョブは番地空間の 8K から開始してロードされる． 0 から 8K はオペレーティング・システムの番地空間の部分である．番地空間が重複していると，番地写像のほかに追加の保護機構が必要となる．これはロックとキーの機構で行なうことができる．別の方法としては，写像のハードウェアを拡張し，各ページ表の欄の空きビットを使用してこのページに対する読出しや書込みが許されるか否かを表示することである．

3.5.3 利　　点

ページングによる記憶管理方式は断片化を解消し，より多くのジョブの番地空間に同時に領域を与えることができる．これは多重プログラミングの多重度を大きくし，記憶装置と処理装置の利用率を高める．再配置型分割割付け方式で生じた詰め合せのオーバヘッドもなくなる．

3.5.4 欠　　点

幾つかの欠点を考慮しておかなければならない．：

1. ページの番地写像のハードウェアは，通常，計算機の経費を増大させ，

同時に処理装置の速度を低下させる.

2. 幾種類かの表――大きなものはページ・マップ表――を格納するために記憶域が必要である. これらの表を保持したり更新するためのオーバヘッドが増大する.
3. 断片化は解消されたが, 同様な現象である**内部的な断片化**, すなわち**ページ余り**が発生する. ジョブは記憶装置のブロック全体を割付けられる. ジョブが5Kバイトを要求し, ブロックの大きさが4Kであったとすると, 2ブロックを割付ける必要がある (すなわち, 8K割付けられるが5Kだけが必要とされている). 平均的には, 各ジョブについて半ページが無駄となっている. ページ余りによる無駄を小さくするためには小さいページの方がよく, ページ・マップ表の欄の個数を減ずるためには大きいページの方がよい. この関係を調整しなければならない. 平均的なジョブの容量を知ることがこの矛盾を解決するのに重要である.
4. 実行しようとするジョブの番地空間を収容するのに十分なだけのブロックがないときには, 依然として記憶域の一部は未使用のままとなる. たとえば, 2ブロックで合計8Kバイトが使用できるとしても, 実行しようとするジョブの最小容量が8Kバイト以上であれば, その領域を使用することができない. この無駄となっている領域は, ページ余りとは別に, 平均的なジョブの番地空間の大きさのおおよそ半分に等しい.
5. 最後に, 前述の記憶管理技法と同様に, 記憶域には滅多に使用されることのない情報まで格納されていることが多い. さらに, ジョブの番地空間の大きさは物理的な記憶装置の容量で制限される.

3.6 デマンド・ページングによる記憶管理

これまで述べてきた方式ではすべて, ジョブの番地空間の全体をロードするのに十分な記憶域がなければ, そのジョブを実行することは不可能であった. この制限のために, ロードして実行するのを待っているジョブが存在していても, 未使用の空き領域が残ることがあった. さらに, ジョブの番地空間を小さくするようにプログラマに圧力をかけることはソフトウェアの生産性を低下させることが多かった. 幾つかの研究の結果によると, ジョブを大きさが限定された番地空間で設計しなければならないならば, プログラミングの経費は大幅に増大することがわかっている (Boehm, 1973).

これらの問題は極端に容量の大きい記憶装置を用いても解決できる. しかし

単純ではあるが，今日ではこの手法は経済的ではない．別の手法は，きわめて容量の大きい記憶域があるような錯覚を与えるオペレーティング・システムを使用することである．この大容量の記憶域は錯覚にすぎないことから，**仮想記憶** (Denning, 1970) と呼ばれる．仮想記憶の技法には主として次の2つがある：

デマンド・ページングによる記憶管理
セグメンテーションによる記憶管理

セグメンテーションによる記憶管理は第3.7節で論じる．

前述の記憶管理技法は，記憶装置の利用率を100%にしようとしたのであるが，デマンド・ページングの記憶管理は論理的にいうと100%以上の利用率を達成することができる．すなわち，多重プログラミングされているジョブの番地空間の総和は物理的な記憶装置の容量を越えることがある．この離れ業は，ジョブの番地空間を完全に主記憶に格納せずに，一部分だけをロードすればよいようにすることによって実現される．

図 3.15 では3つのジョブが記憶装置にロードされ，1,000 バイトのブロックが2つ未使用である．4,000 バイトを必要とする第4のジョブをどのように処理すればよいであろうか（すなわち，ジョブは4ブロック必要とするが，2ブロックだけが使用可能である)．これは 図 3.21 に示したように，ジョブ4の2つのページだけをロードして行なうことができる．ジョブ4は，これら2ページの中の情報だけを参照している限り正しく実行するであろう（たとえば，ジョブ4の番地空間の中の 100 番地にある命令 LOAD 1,1120 は正しく動作する)．

ジョブの番地空間全体をロードせずに実行しようとすることは合理的なことであろうか．そのとおりである．その理由は，ジョブの実行の間では，番地空間の小部分が使用されるだけであるからである．一部しか使用されない理由は次のようなものである：

1. ユーザの作成した誤り処理ルーチンは，データや計算に誤りが発生したときだけ使用される（滅多にないことである)．
2. プログラムの選択機能の内である種のものは互に排他的であったり，毎回の実行には必要とされないものである（たとえば，名前や従業員番号によって分類すること)．
3. 多くの表は，一部しか使用されなくても，番地空間の固定領域を割付けている（たとえば，アセンブラの記号表が1,000個の記号を処理する

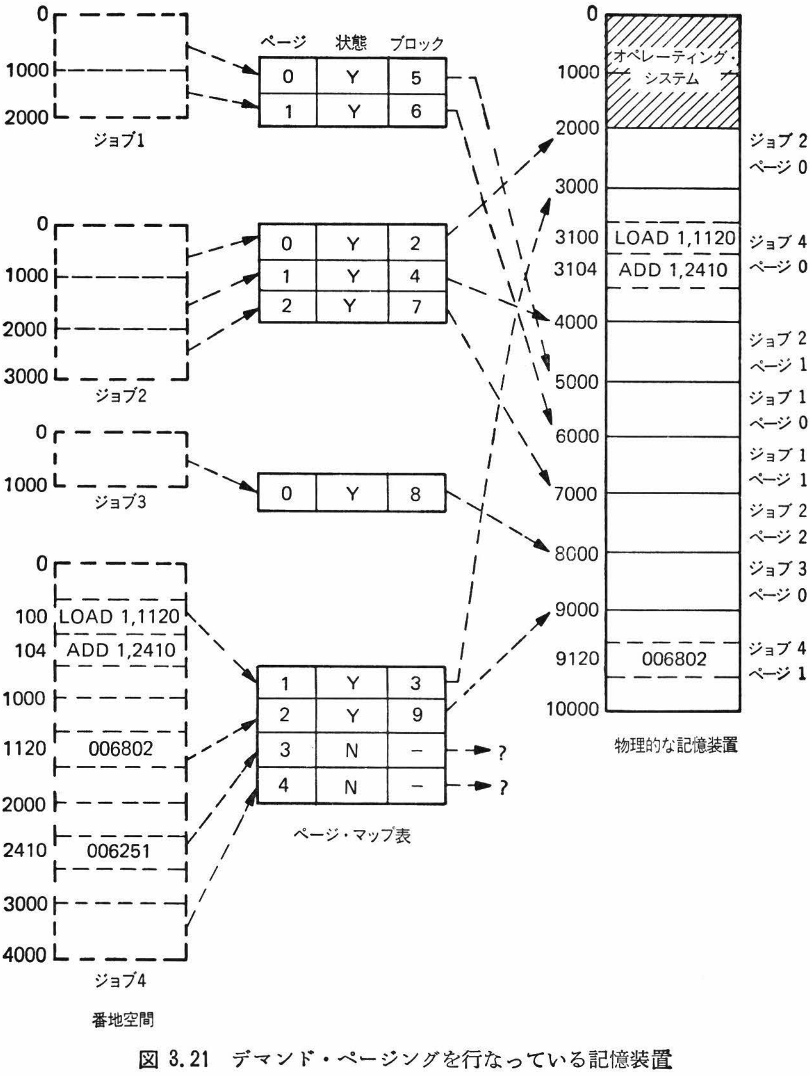


図 3.21 デマンド・ページングを行なっている記憶装置

ように設定されており，ある実行において100個の記号だけが使用されたとすると，表の90%は使用されない)．

4. 共通的に使用されるサブルーチンの多くは，排他的な期間に使用される（たとえば，入力段階のルーチン，計算段階のルーチン，そして最後に，出力段階のルーチン)．これらのルーチンを全部使用するとしても，同時にこれらを物理的な記憶域にロードする必要はない．

ここで次の2つの疑問に答えておかねばならない：

1. 物理的な記憶域に入っていない番地空間の領域をジョブが参照したときにはどのようにすればよいか．
2. どのページを主記憶に残しておくかをどうして決定したらよいか．

図 3.21 では，ジョブ4の番地104の命令 ADD 1,2410 は主記憶中にないページを参照している．このような場合を処理するために，ページ・マップ表のハードウェアを拡張して1ビットの状態ビットを加える（すなわち，Yならば参照可能．Nならば参照不能)．

番地写像のハードウェアが状態＝Nのページ表の欄に出会ったときには，**ページ割込み**を発生する．この割込みはユーザ・プログラムの誤りではないという点で，保護割込みとは違いがあることに注意してほしい．オペレーティング・システムは，この割込みの処理として要求されたページをロードし，対応するページ表の欄を設定しなければならない．その後に，命令の再実行が可能となる．このページは“要求（デマンド）によってロードされた”という（すなわち，ジョブ4の番地空間のページ2は，ページ割込みによってそのページがジョブに必要となったことを通知されたので，ロードされた)．このようなわけで，本方式は**デマンド・ページングによる記憶管理**と呼ばれる．特に，最初ジョブの実行がスケジュールされるときには，通常，実際には第1ページだけがロードされる．ジョブで必要となる他のページはすべてその後で必要とされたときロードされる．このようにすることによって，不必要なページをロードしないようにすることができる．

要求されたとき，これらのページにどこからロードされるのであろうか．ジョブの全番地空間の写しは2次記憶装置（たとえば，ディスクやドラム）に格納されている．ページが主記憶に必要となると，2次記憶装置から読出される．

デマンド・ページングによる記憶管理技法は，番地空間の大きさが物理的な記憶装置の容量に制約されないので，きわめて魅力的である．しかし，システ

ムは幾分複雑である．図 3.21 でジョブ4の 104 番地の命令 ADD 1,2410 に達したとき，ページ2を物理的な記憶装置の中にロードしなければならない．記憶ブロックはすべて使用中であるから，ページ2をどこに置けばよいであろうか．このためには，**ページ・スワッピング**とか，**ページ置換え**，**ページ交換**などと様々な名前で呼ばれている技法が必要となる．

いったん記憶装置がページで一杯になったならば，まず現在記憶装置に入っているページを取除いてからでないと，別のページをロードすることができない．**置換えられる**ページは，新しいページがロードされる前に2次記憶に写し返される（すなわち，2つのページが主記憶と2次記憶の間で交換される）．どのページを記憶装置から取除けばよいかをどのようにして決定すればよいであろうか．置換えアルゴリズムは多くの研究の主題となっている (Belady, 1966; Coffman, 1968；Denning, 1970, 1972；Gustavson, 1968；Randell, 1968)（第 3.6.2 節参照）．

主記憶からページを取除いた直後に，そのページが参照されてページ割出しが起こり，そのページが必要になるようなことはきわめて運の悪いことである．主記憶と2次記憶の間で過度にページを移動させる現象は，計算機の精力を消耗するが役に立つ仕事はほとんど実行しないことから**スラッシング**（**無理押し**）と呼ばれる．スラッシングとその予防は研究者の大きな注目を受けている (Denning, 1968, 1970)．オペレーティング・システムのソフトウェアがページ置換えを決定するのを助けるために，ハードウェアは，通常，ページの使用頻度に関する情報を保存している．このようなハードウェアに関係する情報の例は第 3.6.1 節で示されている．

デマンド・ページングは，IBM 370 の VS 1 や VS 2, Honeywell 6180 の MULTICS, UNIVAC 70/46 シリーズの VMOS などの，今日の多くのオペレーティング・システムで採用されている．

ページングを行なう記憶管理の4つの機能は一層複雑で，一層柔軟となっている：

1. 状態の把握——これは3つの表によって行なわれる．
   a．ページ・マップ表——番地空間に1つずつ
   b．記憶ブロック表——システムに1つ
   c．ファイル・マップ表——番地空間に1つずつ
2. 記憶域をどのプロセスにいつ与えるかの方針——一部はジョブ・スケジューラにより決定されるが，動的には，デマンド・ページ割込みによっても決定される．

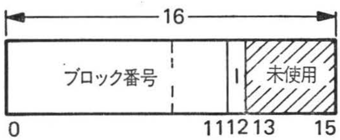
3. 割付け——ブロックを割付ける必要があるときには，ブロックを探して，状態を変更しなければならない．
4. 解　放——割付けのために使用できるブロックが見つからないときには，割付けられている記憶ブロックを解放したり割付けを変更したりする必要がある．ジョブが終了したときには，そのジョブが使用していたブロックはすべて使用可能となる．

3.6.1 ハードウェア

第3.5節で述べたように，デマンド・ページングのためにはページ・マップ表を用いる番地写像のハードウェアが必要である．このハードウェアには3つの点で追加が必要である：

1. ページが主記憶に入っているか2次記憶に入っているかを示すページ・マップ表の状態ビット
2. ジョブが主記憶に入っていないページをアクセスしようとしたとき，制御をオペレーティング・システムに移すための割込み動作
3. どのページを取除くかをオペレーティング・システムが決定するための補助として，必要ならば個々のページの使用頻度の記録

これらの要求は IBM 370 では次のようにして満たされている（拡張制御モード，すなわち，番地変換が行なわれているとき）[1]．第 3.5.1.2 節で述べたページ・マップ表の欄は変更されて，以前には使用されていなかったビットを使用して，この欄が参照されたとき割込みを発生するか否かの表示を行なう（I＝1は割込みを発生することを意味する）．



ページ割込み（**ページ割出し**とも呼ばれる）条件が検出されると，符号＝17のプログラム割込みが発生する．加えて，参照しようとした論理番地が記憶装

1) PSW のEビット（ビット 12）が1のとき拡張制御モードが有効となる．IBM 370 では 2K または 4K のページのハードウェアを持っている．動作モード（2K または 4K）は制御レジスタ 0 のビット 8 および 9 を設定することによって決定される．本章では 4K モードの場合のみを論じる．

置の 144 番地の 1 語に格納される．この語の内容によって，オペレーティング・システムはどのページをロードすればよいかを決定することができる．

各記憶ブロックにはもう 2 ビットが関連する．（これらの 2 ビットは，実際には 2K の記憶ブロックに関するロック・レジスタの一部として格納される．）**参照ビット**はこのブロックが参照されたとき（すなわち，ロード，格納，実行）1 に設定される．**変更ビット**はこのブロックの中の番地が変更されたとき（すなわち，格納）1 に設定される．オペレーティング・システムは特権命令である記憶キー設定命令 SSK を用いてこれらのビットを帰零することができる．Honeywell 6180 などの別のハードウェアにおいては，これらのビットはページ・マップ表の欄に格納されている．

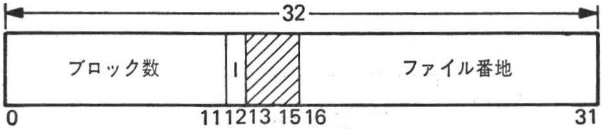
オペレーティング・システムの補助のために，他の機能がハードウェアとして装備されることがある．たとえば，通常，オペレーティング・システムは**絶対モード**とか**基本制御モード**（IBM 360 のように動的な番地写像を行なわない）で実行される．番地写像は，仮想番地を対応する物理的な記憶番地に変換する"実番地ロード"（LRA）命令などを使用して有効にすることができる．そのページが主記憶の中になければ，割込みを発生せずに特別の条件符号が設定される．この機能は，次節において論じるように，入出力要求の処理において重要である．

3.6.2 ソフトウェア

デマンド・ページングはオペレーティング・システムにとってすばらしい柔軟性を提供する．したがって，大変注目を集めている．本節では，主要なアルゴリズムに限って述べる．本節に含まれていない問題については文献を参照してほしい．

3.6.2.1 ファイル・マップの必要性

デマンド・ページングによる記憶管理は，2 次記憶装置にジョブの番地空間の写しを作ったり参照したりするために情報管理と関係する必要がある（情報管理については第 6 章で論じる）．ここでは，各ページは 2 次記憶装置の番地を一意的に割付けられていると仮定する．この番地は**ファイル番地**と呼ばれる．この番地は論理的にページ・マップ表の各欄の一部とみなされる．ファイル番地が 16 ビットの長さのときは次のようになる：



ページ・マップ表は番地写像のハードウェアと直接関係しているが，ファイル番地（および，これに関連する状態情報）はソフトウェアだけで処理され，その書式は可変である．したがってファイルの情報は，通常，ファイル・マップ表と呼ばれる別個の表に格納される．この表はハードウェアによって使用されることはない．

ファイル・マップ表とページ・マップ表，および記憶ブロック表の関係は図 3.22 に示されている．これらの表はすべて，オペレーティング・システムの領域として主記憶に常駐し，各番地空間に別個のページ・マップ表とファイル・マップ表とが存在する．図ではジョブ 2 は 3 ページの内 2 ページだけが主記憶に入っている．3 ページ分の写しはすべて 2 次記憶装置に格納されている．

3.6.2.2 ページ割込み処理概観

前述の記憶管理方式では，記憶域割付けは完全に静的に行なわれた．すなわち，ジョブが開始，あるいは終了したときだけ，記憶域の割付けや解放が行なわれた．デマンド・ページングにおいては，ジョブ実行中に記憶域を割付けたり解放したりする必要がある．記憶管理のルーチンはページ割込みの手段を用いて呼出される．

デマンド・ページングでは，ハードウェアとソフトウェアの間に密接な相互関係が存在する．この関係は図 3.23 に簡単に示されている．この流れ図の最初の部分は最も頻繁に実行される部分であり，番地写像のハードウェアの一部として組込まれる．第 2 の部分はオペレーティング・システムの中の割込み処理ルーチンとして組込まれる．研究実験用のシステムにおいては，この流れ図のもっと多くの部分がハードウェアとして組込まれているものもある (Liskov, 1972)．将来のシステムにおいては，オペレーティング・システムはより一層ハードウェアと密接に結合することになるであろう．

図 3.23 の流れ図において，“ページは主記憶に入っているか” という問には，ページ・マップ表の欄にあるページ割込みビットの状態を用いて答えていた（すなわち，0 は入っていること，1 は入っていないことを意味する）．同じ検査は命令の中のデータ番地のすべてと（たとえば，文字移動命令（MVC）

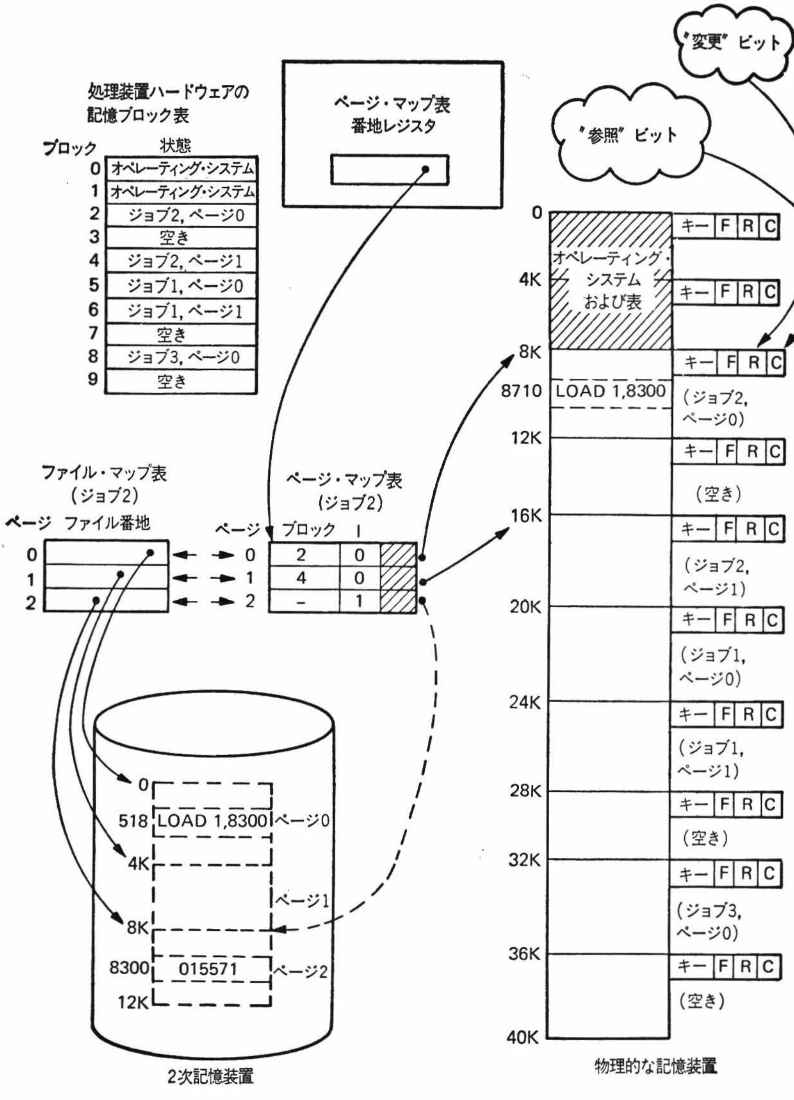


図 3.22 ファイル・マップ表とその他の表の関係

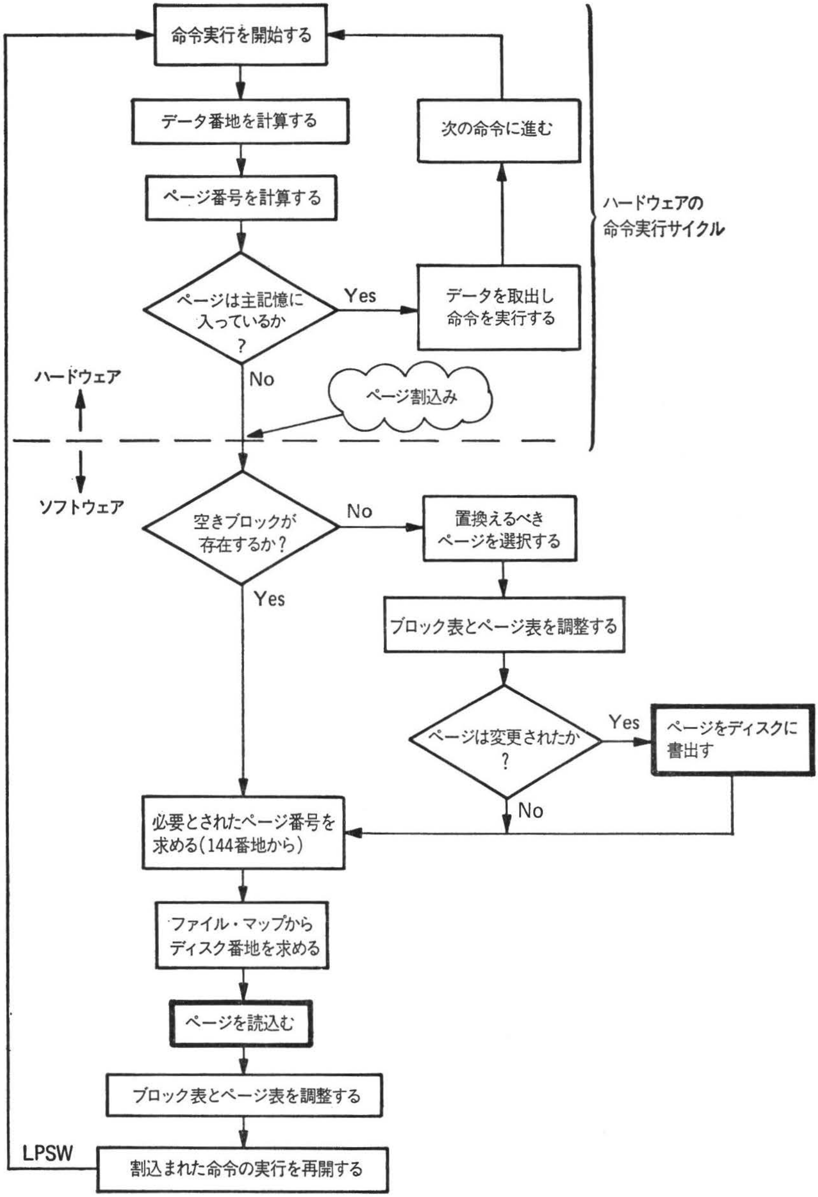


図 3.23 ハードウェアとソフトウェアの相互関係

は2つのデータ領域を参照する）命令自身について行なう必要がある．実際IBM 370では，1つの命令を実行するのに8回のページ割込みが起こることがある．

図3.24はこれを示している．12286番地にある文字移動命令（MVC）は4バイトを占めている（12286～12289）．この命令の半分はページ2の中にあり，残りの半分はページ3の中にある．両方のページは主記憶中になければならない．さらに，移動すべきデータ部分AおよびBも2ページにまたがっていることもある．したがってこれらの6ページはすべて主記憶の中になければならない．文字移動命令が4094番地に示したような代行命令（EX）によって要求された場合には，さらに2ページが主記憶中に必要となる，これらのページがどれも主記憶中にない場合には，8回のページ割込みが発生する．IBM 370の長い文字移動命令（MVCL）や，多重の間接番地指定を行なう他の計算機の場合には，無限回のページ割込みが発生する可能性もある．

図3.23の流れ図に戻ろう．“空いているブロックが存在するか”という問は

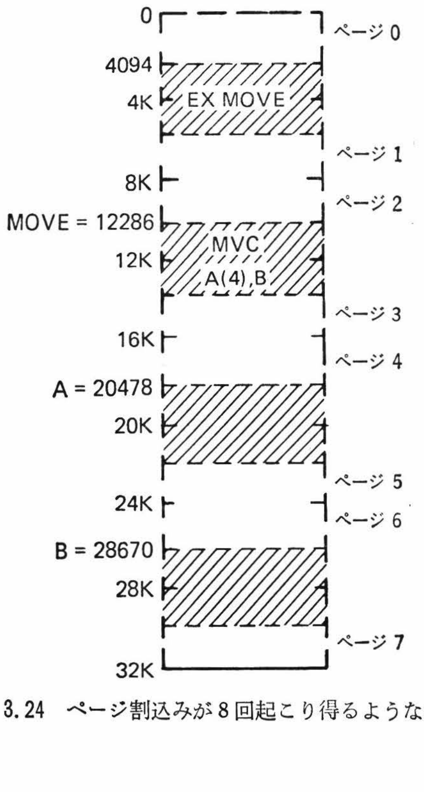


図 3.24 ページ割込みが8回起こり得るような場合

記憶ブロック表で空きブロックを探すことによって答えることができる．空きブロックがあれば，144番地の割込み情報の語に入っているページ番号が取出され，ファイル・マップ表から，そのページの2次記憶装置における番地が求められる．ページが主記憶に複写された後に，ページ・マップ表と記憶ブロック表とを変更しておかねばならない．その後で命令の再実行が可能となる．空きブロックが存在しない場合には，主記憶からページを1つ取除かなければならない．ページを選択した後に，“変更”ビットが検査される．変化していないならば，2次記憶装置にあるページの写しは同一のものであるので，ページを2次記憶装置に書出しておく必要はない．そうでなければ，ページは書出される．その後でこの記憶ブロックは他に使用することができるようになる．

上述したように，ページ割込みを処理するには，ページを読込んだり書出したりするために1～2回の入出力動作（**ページ入出力**と呼ばれる）が必要となる．処理装置がページ入出力の完了を待合せてアイドルとなる時間を最小とするために，利用者ジョブから陽に入出力動作が行なわれたときと全く同様に，多重プログラミングが行なわれることが多い．しかし，多重プログラミングを行なうことはシステムを複雑にする傾向がある．

その1つの例として，割付け／空きビットに加えて，記憶ブロックの状態ビットが別に必要となる．この新しい状態は**遷移中**と呼ばれる．ジョブ1のためにページをロード中で，処理装置がジョブ2に割付けられている状況を考えてみよう．ジョブ2は直ちにページ割込みを起こし，記憶ブロックを要求する．ジョブ1のブロックに“割付け状態”の表示があるとページ割込みハンドラはこのブロックを取除こうとすることがある．しかし，ページはまだロードすらされていないのである．ブロックに“使用可”の表示がしてあれば，ジョブ2に割付けられるであろう．このような場合にはジョブ1のページとジョブ2のページが同一のブロックにロードされることになる．“遷移中”の表示は問題のブロックをこの時点で割付けの対象としてはならないことを意味している．ページ入出力が完了したときには，状態は割付けあるいは使用可に，状況に応じて変更される．

多くのシステムではページ入出力の回数が非常に大きくなることがある．このような場合には，高速度の2次記憶装置と複雑な装置管理技法が採用される．これらの話題については第5章において論じる．

3.6.2.3 ページ置換えのアルゴリズム

図 3.23 の流れ図でただ1つ明瞭でなかったのは“取除きのためにページを

選択する”手順であった．この選択はどのようにして行なわれるのであろうか．記憶ブロック3を占めているページを取除くことにしておくと，この手順はきわめて簡単に実行することができる（ブロック3はオペレーティング・システムの領域の直後のブロックである）．しかし，この技法は簡単ではあるが危険でもある．図 3.24 の場合には，1つの命令の実行に8回の割込みが起こった．各ページ割込みはロードしたばかりのページを取除くこととなり，8ページが同時にロードされることはなくなる．したがって，この命令は決して実行されることはない．これは前述したスラッシングの現象の極端な場合である．

取除くのに最善のページとは何であろうか．直観的には，それは再び必要とならないページ，あるいは，少なくとも当分の間必要とならないページである．このようなアルゴリズムは最小（MIN；Belady, 1969）あるいは最適（OPT；Mattson, 1970）アルゴリズムと呼ばれて研究されている．不幸にして，これらのアルゴリズムでは，未来に起こるページ割込みに関する知識があらかじめ必要である（これは株式投資のアルゴリズムでどの株を買えばよいか――“それは最も価格が上昇するものである”――というのと同様である）．したがって，最小あるいは最適アルゴリズムの解は理論的な値である．これらの値は，実現可能のアルゴリズム（すなわち，未来の知識を必要としないアルゴリズム）の有効性を測る尺度として使用できる．

置換えアルゴリズムには多くの方式が存在する．ページングにおける置換えの問題は理論的に興味のある領域である．本節では次の話題について論じる：

1. FIFO（先入-先出，first-in first-out）方式
2. LRU（最近最も使用されていない）方式
3. 項の結合（tuple-coupling）

置換えの背後にある基本的な考え方と問題点はたとえを用いて説明することができる．読者にはスーパマーケットの経営の責任があるものとしよう．現時点で商品の棚が一杯になっている．新しい製品が導入されてそれを売出したい．この商品をどこに置けばよいであろうか．明らかに，すでに棚にのっている商品を置換える必要がある．どの商品を選べばよいかが問題である．読者には店員に与えるアルゴリズムを考案することができるか．

1つのアルゴリズムは，最も長い期間棚にのっていた商品を置換えることであろう（流行遅れになっている可能性がある）．このアルゴリズムは FIFO のアルゴリズムに対応する．在庫管理を行なう店員にとって FIFO はよいアルゴ

リズムであろうか．恐らくそうではないであろう．置換えられる商品としては，砂糖や紅茶といった，50年間売り続けてきたものが選ばれるであろう．

店員にとってもっとよいアルゴリズムは，店を回って商品のほこりを払い，最もほこりをかぶっている商品，すなわち，売れていず，手を触れたことのない商品を置換える．これは LRU アルゴリズムに対応する．

置換えアルゴリズムの有効性の尺度は，置換えた商品を取りに地下室へ降りて行く必要回数を用いることができる．

置換える商品の量を半分だけにしたら（ページの大きさを半分にする），地下室に行く回数（ページングのアルゴリズム）にどのような影響が出るであろうか．すなわち，すべてのゼリーを置換えずに，木いちごとオレンジとレモンのゼリーだけを置換える．このようにすると，すべての商品を置換えたときに生じたのに比べて，取りに行く回数が大幅に減少するであろう．しかし，このような減少は，滅多に触れられることのない商品を置換えたときだけに起こるであろう．木いちごとオレンジとレモンだけを売っていたとしたらそのようにはならなかったであろう．半分だけを置換えるのに比べて，すべてのゼリーを置換えたときには何回余計に地下室へ取りに行かねばならなかったであろうか．直観的には，その回数は多くて2倍だと考えるかもしれない．しかし，この回数は2倍以上であることが証明されている（Madnick, 1973)．

最悪の場合を限定する方法はあるであろうか．これは採用するページング・アルゴリズムに1つの特性を追加して行なうことができる．この性質は項の結合と呼ばれ，最悪の場合でも，ページの大きさを半分にしなかったときに起こる回数が2倍以上にならないことを保証する．

スーパマーケットの繁栄を祈って計算機に戻ろう．

3.6.2.4 ページ置換えの効率とプログラム動作のモデル

ページ置換えアルゴリズムの効率を評価するためには効率のモデルを使用する必要がある．本節では簡単ではあるが役に立つモデルを示そう．ページ置換えアルゴリズムの効率は実行するプログラムに関係するから，**プログラムの動作**を調べておく必要がある．

本節ではプログラムを，まず実行の軌跡，番地の軌跡，そして最後にページの軌跡を求めて抽象化する（図 3.25 参照)．**実行の軌跡**はプログラムで実行される命令をその順番に並べた一覧表である．**番地の軌跡**は各命令の実行で必要となった番地の表である．**ページの軌跡**は番地の軌跡で必要となった対応ページの表である．ページの軌跡はプログラム実行を特徴づけるのに十分なデータ

実行の軌跡			番地の軌跡	ページの軌跡 (ページ容量=1000バイト)
100	LOAD	1,1120	100	0
			1120	1
104	ADD	1,2410	104	0
			2410	2
108	STORE	1,1124	108	0
			1124	1
112	MOVE	1140(4),	112	0
		3998	1140	1
			3998	3,4 [3998, 3999, 4000, 4001]
116	HALT		112	0

図 3.25 実行の軌跡，番地の軌跡およびページの軌跡

である．図 3.25 の命令 ADD 1,2410 が命令 SUB 1,2464 に変更されても，ページ 0 およびページ 2 は依然必要である．ページの要求だけが重要なのであり，命令の種類が重要なのではない．

本節の簡単な効率のモデルは 3 つのパラメータを使用している：

1. ページの軌跡 P
2. 記憶装置の容量 M
3. 置換えアルゴリズム R

FIFO の置換えを行なった簡単な例が図 3.26 に示されている．ページの軌跡は表の上に示されている．この例では，記憶容量はわずか 3 ブロック $M=3$ である．ページが主記憶にロードされたときには，ページ番号が主記憶中のページの一覧表に追加され，F の行に印“+”がつけられる．M の列は各時点での主記憶中のページを示している．

主記憶の中にないページに対して参照が行なわれたとき，この事象を**ページ取出し**あるいは**ページなし**の状態であるという．たとえば，第 3 のページ参照はページ 2 に対するものであるが，主記憶の中にはページ 3 とページ 4 だけが入っている．したがって，ページ取出しが要求される．図 3.28 の 8 番目のページ参照のように必要なページが主記憶に入っているとき（ページ 4 が必要である．主記憶にはページ 5，3，および 4 が入っている)，この事象はページありと呼ばれ，ページのローディングは不必要である．

時間	1	2	3	4	5	6	7	8	9	10	11	12
P =	4	3	2	1	4	3	5	4	3	2	1	5

M = 3		4+	3+	2+	1+	4+	3+	5+	5	5	2+	1+	1
			4	3	2	1	4	3	3	3	5	2	2
				④	③	②	①	4	4	④	③	5	5
F =		+	+	+	+	+	+	+			+	+	

F = 9

f = 9/12 = 75%

図 3.26 FIFO 方式のページ軌跡の解析

効率は，通常，**成功関数** S あるいは**失敗関数** F によって測定される．これらはそれぞれページあり，あるいはページなしの回数である．$|P|$ をページの軌跡 P におけるページ参照の回数とすると，$S+F=|P|$ である．別の有用な効率の測度は，成効率 $s=\dfrac{S}{|P|}$，あるいは失敗率 $f=\dfrac{F}{|P|}$ である．なお，$f=1-s$ である．図 3.26 の例においては 12 回の参照で 9 回ページなしがあった．したがって失敗率 f は 75% である．ずっと記憶ブロック数の大きく，もっとページの軌跡の長い実際のシステムにおいては，通常 f は 1 %以下である．

番地写像のハードウェアがあるので，ページの主記憶中の物理的な位置は重要ではない．置換えアルゴリズムをシミュレートするのに，ページ軌跡の解析の列に入っているページの順番を利用することができる．たとえば，FIFO の置換えでは，ページを置換える必要のあるときには最も長期間主記憶に入っているページを選択したい．図 3.26 で列は“最も古い”ページがつねに底に，“最も新しい”ページが頂上に並べられている．置換えアルゴリズムでは，図 3.26 で円で囲んだものが選択される．実際の FIFO の置換えの組込みにおいては，ページがロードされた時刻を格納することが可能で，置換えのためには，“最も古い”時刻の探索を行なうことができる．順番に並べた一覧表を管理しておく方が効率がよい．これはポインタを使用するもので，長たらしい探索を行なわずに図 3.27 のようにして置換えるべきページを選択することができる．

図 3.27 は記憶ブロック表（前に図 3.18 で論じた）の使用を拡張した順番

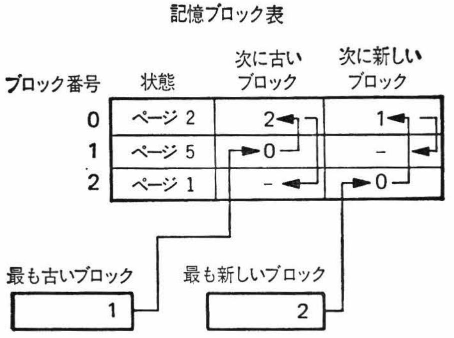


図 3.27 ページ置換えのための記録の管理

づけの機構である．各記憶ブロックの欄は“次に古い”欄を指している．1つの変数を使用して“最も古い”欄を表示し，これによって鎖の開始点とし，かつ，次に置換えるページを指定している．ページが主記憶にロードされると，このブロックは“最も古い”鎖の末尾，および，“最も新しい”鎖の先頭に置かれる．図 3.27 の例は図 3.26 の最終的な順番を示し，ページ1，ページ2，およびページ5が“最も新しい”から“最も古い”の順に並んでいる．

3.6.2.5 FIFO の置換え

FIFO の置換えは，前節にて述べたように，組込むのには単純なアルゴリズムであるが，ページをロードした直後にそれを置換えるという可能性を解消している．

FIFO には3つの欠点がある：

1. ページが頻繁にたえず使用されていても，いつかは最も古くなって置換えられてしまう．置換えられた直後に必要となるかもしれない．
2. ある種の奇珍な副作用が起こり得る．
3. もっと有効なアルゴリズムが見つかっている．

FIFO 異常と呼ばれる副作用が最もよく注目されている (Belady, 1969)．これはある条件の下では物理的な記憶容量を増すほど効率が低下するというものである．図 3.28 は図 3.26 と同じページの軌跡を示しているがもっと記憶容量が大きく $M=4$ である．

記憶容量が大きく，直観的には記憶容量が大きいほど効率もよいと考えられるにもかかわらず，失敗率関数は 75% から 83% に上昇している．

	1	2	3	4	5	6	7	8	9	10	11	12
P =	4	3	2	1	4	3	5	4	3	2	1	5
M = 4	4+	3+	2+	1+	1	1	5+	4+	3+	2+	1+	5+
		4	3	2	2	2	1	5	4	3	2	1
			4	3	3	3	2	1	5	4	3	2
				4	4	④	③	②	①	⑤	④	3
F =	+	+	+	+			+	+	+	+	+	+

$F = 10$

$f = \frac{10}{12} \approx 83\%$

図 3.28 FIFO 異状の例

このような異常を示すページ軌跡は，もちろん，実際には滅多に起こらない．にもかかわらず，この不都合な現象およびその他の欠点のために，FIFO はよい方法とはみなされていない．

3.6.2.6 LRU の置換え

多種のページ置換えアルゴリズムが提案され試験されている．最もよく知られているのが **LRU**（最近最も使用されていないもの）と呼ばれる技法である．この技法は最も長い期間参照されなかったページを選んで置換えようとするものである．これに対して，FIFO は参照の回数や時期に無関係に，最も長期間主記憶に入っているページを置換えようとするものである．LRU は，ページが参照されればすぐまた参照される確率が大きいという見解に基づいている．逆に，長い間参照されなかったならば，近い将来に必要となる確率は小さいと考えられる．これが第 3.6.3 節で論じる**局所性の原理**の基礎ともなっている．

LRU のページ軌跡の解析では，図 3.29 に示したように，最も最近参照されたページを M 表の最上位におく．最近“最も使用されていない”ページは M 表の底に落ち，置換えのための候補とされる．LRU は多数の理論的に興味ある性質を有していることがわかっている．特に，LRU は，**スタック・アルゴリズム** (Mattson, 1970) と呼ばれる置換え技法のクラスに属するものである．このクラスのアルゴリズムでは，記憶容量を増すときにページ割込みの回数が増加することがない．これは FIFO 異常と対照的である．図 3.29 は，記憶容

	1	2	3	4	5	6	7	8	9	10	11	12
P =	4	3	2	1	4	3	5	4	3	2	1	5
M = 3	4+	3+	2+	1+	4+	3+	5+	4	3	2+	1+	5+
		4	3	2	1	4	3	5	4	3	2	1
			④	③	②	①	4	3	⑤	④	③	2
F =	+	+	+	+	+	+	+			+	+	+

(a) M = 3

F = 10

$f = \frac{10}{12} \approx 83\%$

	1	2	3	4	5	6	7	8	9	10	11	12
P =	4	3	2	1	4	3	5	4	3	2	1	5
M = 4	4+	3+	2+	1+	4	3	5+	4	3	2+	1+	5+
		4	3	2	1	4	3	5	4	3	2	1
			4	3	2	1	4	3	5	4	3	2
				4	3	②	1	1	①	⑤	④	3
F =	+	+	+	+			+			+	+	+

(b) M = 4

F = 8

$f = \frac{8}{12} \approx 67\%$

図 3.29 LRU 置換えアルゴリズムの例

量の2つの値 $M=3$, $M=4$ に対して LRU の置換えを示したものである．各列の上位の3つの要素はつねに同一であることに注目してほしい．たとえば，7番目のページ参照の後で $M=3$ の例ではページ5，3，4を，$M=4$ の例ではページ5，3，4，1を納めている．

$M(P, c, t)$ が主記憶中にあるページの一覧表を表わすとする．ここで P はページの軌跡，c は記憶容量，t は参照時刻とする．図 3.29 の LRU の置換えでは，$M(P, 3, 7)=5, 3, 4$；$M(P, 4, 7)=5, 3, 4, 1$ である．LRU などのスタック・アルゴリズムにおいて，任意の P, c, t の値に対して $M(P, c, t) \subset M(P, c+1, t)$ であることが証明されている (Mattson, 1970)．この関係はスタック・アルゴリズムの**包含関係**と呼ばれている．

包含関係の正式な証明は多少長たらしいので本書では示さない．証明の手がかりは，$M(P,\ c,\ t)$ の値をどのようにして決定するかを考えることにより得られる．たとえば，$M(P,\ 3,\ 7)$ を決定するには7番目のページ参照――ページ5――から開始して，ページの軌跡を逆にたどり，最初に出てきたページ番号を3つ求めればよい．$M(P,\ 3,\ 7)$ は5，3，4である．$M(P,\ 4,\ 7)$ はページ番号を4つ求めればよい．このとき同じページ番号が現われても2回含めないことにする．その例は $M(P,\ 4,\ 9)$ で説明される：

t = 9 8 7 6 5 4

P(t) = 3 4 5 ~~3~~ ~~4~~ 1

したがって，$M(P,\ 4,\ 9)=3,\ 4,\ 5,\ 1$ である．時刻6および5のページ番号3および4はすでに求められているので捨てられる．$M(P,\ c,\ t)$ を求めるには，t 番目のページ参照から始めページ軌跡を逆にたどり，最効に出てくる c 個のページ番号を求める．$M(P,\ c+1,\ t)$ を求めるには，最初に出てくる $c+1$ 個のページ番号を求める．この定義より，$M(P,\ c,\ t) \subset M(P,\ c+1,\ t)$ であることは容易に理解されるであろう．すなわち，$M(P,\ c,\ t)$ に含まれるページ番号は，$M(P,\ c+1,\ t)$ に含まれている．この包含関係は，記憶容量を増加してもページ割込みの発生回数を増加させないことを保証する．

3.6.2.7 LRU の近似

LRU の置換えアルゴリズムは，**ページ参照の度**にページ置換えのための状態情報を更新しなければならない（FIFO の置換えでは，ページが主記憶にロードされたときだけこの更新を行なえばよかった）．このような更新をソフトウェアだけで絶え間なく行なうとしたら絶望的な経費を要するであろう．ハードウェアの LRU 機構は実行効率を低下させ，同時に，大幅に経費を増加させる．これらの問題点によって，真の LRU 置換が採用されることはほとんどなく，代わりに，LRU を近似した簡単で効率のよいアルゴリズムが開発されている．

第3.6.1節でハードウェアを述べ，**参照ビット**が導入された．参照ビットは各記憶ブロックに関連して，そのページが参照されたときハードウェアで1に設定される．このブロックごとに1ビットの参照ビットを使用し，図 3.30 および図 3.31 に示したようにして，LRU 置換えを近似することができる．

このページ置換えのソフトウェアは，参照ビットを定期的に帰零する．一方

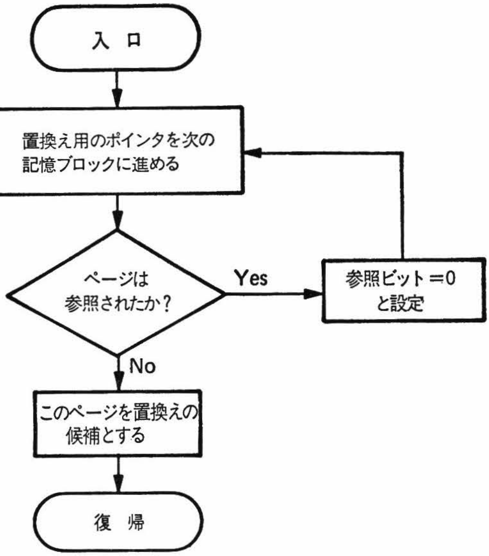


図 3.30 LRU の近似アルゴリズム

利用者ジョブの実行によって幾つかの参照ビットは1に設定される．ある時点で参照ビットが0であったとすると，この参照ビットが0にされてからこのページは参照されたことがないことを意味している．したがって，このページは"最近使用されていない"ので，置換えの候補となる．図 3.30 のアルゴリズムは，参照されていないブロックを探して，記憶ブロック表を循環している．探索の間に，参照されたページの参照ビットを帰零する．

図 3.31 は NUR（最近使用されていない）アルゴリズムを示している．図 **3.31 a** でブロック1（ページ5）は最後にロードされたページである．ページを置換える必要性が生じると，ブロック2から開始して参照ビットを調べる．ブロック3（ページ1）は参照ビットが0であるので置換えのために選択される．同時に0でなかった参照ビットは 図 **3.31 b** のように帰零される（たとえば，ブロック2）．ページ割込みがこの直後に起こったときには次に選択されるブロックは0（ページ4）である．

多数の LRU の近似が使用できる．図 3.30 のようなアルゴリズムは組込みが容易であり，LRU の妥当な近似の1つである．別の幾つかの方法については Corbato (1969) の報告がある．

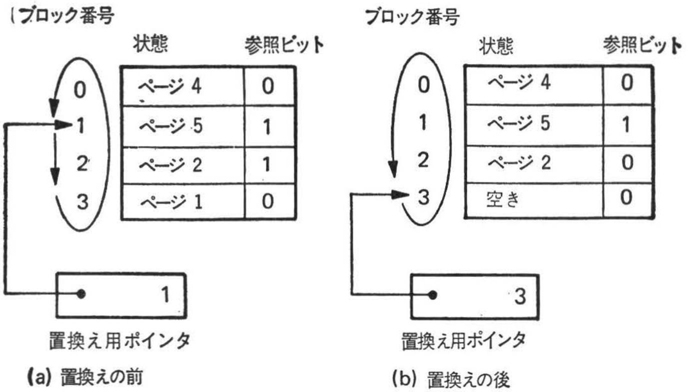


図 3.31 LRU の近似アルゴリズムの例

3.6.2.8 ページ容量に関する異常現象

ページの大きさは，デマンド・ページングを行なう記憶管理システムの重要なパラメータである．ページの大きさは各種の条件の妥協で決定される．一方では，ページ・マップ表の容量を小さくし，2次記憶装置の大きなアクセスと読出しの時間を補償するだけの大きなページを使用したい（1バイトのページのアクセスと読出しは5ミリ秒かかるが，4096バイトのページは7～8ミリ秒で取出すことができ，時間にしてわずか50～60%の増加である）．他方では，ページ余りを小さくしたいので，小さなページがよい．たいていの計算システムでは，1Kから4Kバイトのかなり大きなページが一般的である（大きな例外については第3.10.3節で論じるキャッシュ・システムを参照）．別の重要な問題点はページ割込み回数の影響である．

ページの大きさを半分にしたときの影響を考えてみよう．元のページ軌跡でページ p が参照されていると，容量が半分のページの軌跡では，ページのどちらの半分に入っていたかによって，ページ p' または p'' が参照される．

ページの大きさを N および $N/2$ バイトとしよう．物理的な記憶容量が $2N$ バイトのときは，大きな2ページあるいは小さな4ページが格納できる．図3.32はLRU置換えによって2つのページの大きさでページの軌跡を求めたものである．この例では，大きいページ p の片一方 p' だけが実際に必要となっている．小さいページの方がページ割込みの回数が少ない．比 $r=Fa/Fb=0.5$ は，ページの大きさが小さいと割込み回数が半分になっていることを示している．この現象は，ページが小さいときには大きなページの中の不要なバイトをロードする必要がないということから，圧縮と呼ばれている（Denning,

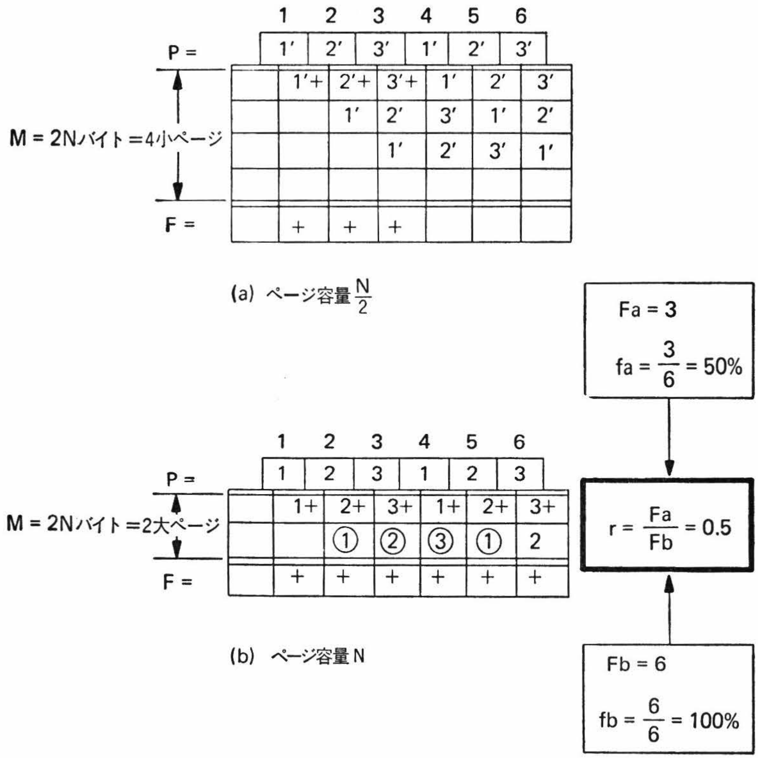


図 3.32 小さいページ容量の利点 ($Fa/Fb=0.5$)

1970).

不幸にして，圧縮はつねに起こるとは限らない．図 3.33 では大きいページの両方の部分が実際に必要となっている．このようなページをロードするには，ページの大きさを半分にすると，2倍のページ割込みが起こる．この場合，比 $r=Fa/Fb=2$ は，ページの大きさが小さいと不利益であることを示す．

実際のプログラムは，はるかに長いページの軌跡を有し，図 3.32 と図 3.33 の両方の性質を示す．システムのページの大きさを決定するに当たって，設計者は可能な限り妥当な大きさを選ぶようにしなければならない．極端な場合である効率の限界を知っておくことは有用である．小さいページに都合のよい図 3.32 の場合には，比 $r=Fa/Fb$ は，1′，2′，3′ の繰返しによってページの軌跡を長くすると，0 にきわめて近くすることができる．たとえば，次のページ軌

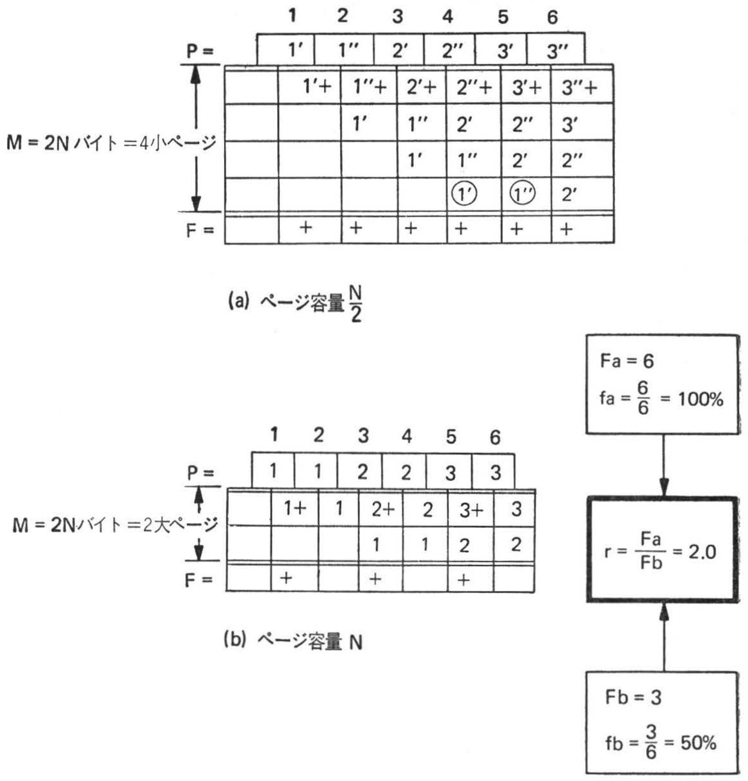


図 3.33　大きいページ容量の例（$Fa/Fb=2.0$）

跡では，比 $r=\frac{3}{12}=0.25$ である：

$$P=1',2',3',1',2',3',1',2',3',1',2',3'$$

一般に，このパターンの長さが p のページ軌跡では，比 $r=\frac{3}{p}$ である．p が大きいとき，$r\approx 0$ である．

図 3.33 は小さいページの場合であるが，大きいページの 2 つの部分はすべて必要となるので，極端に悪い場合と考えられる．この場合は比 $r=2$ である．もっと悪い $r=2.2$ の場合が 図 3.34 に示されている．$r=2$ が最悪の場合でないとしたら，最悪の場合はどうであろうか．LRU の置換えを採用した場合，効率の比 $r=Fa/Fb$ が漸近的に局限値 $\frac{M}{N}+1$ に近づくようなページ軌跡が存

在することが証明されている(Madnick, 1973). したがって, たとえば, 容量 M が 1024K バイトの記憶域を持ち, ページの大きさ N が 4K バイトのデマンド・ページングを行なっているシステムで, ページの大きさを 2K バイトに減じたとき, 増加量は $\frac{1024\text{K}}{4\text{K}}+1=257$ 倍となることがある (すなわち, ページ割込みは 25, 600% 増加する).

3.6.2.9 項結合法によるページ置換え

ページ容量の小さい方がページ軌跡の点でも好都合であると期待できるので, アクセス時間の小さい高性能な2次記憶装置の発達によって, ページ容量を小さくすることが有利である. ページ容量が小さいと不都合な場合の限界が明らかにされているならば一層都合がよい. 比 r の値が2以上とならない保証のある**項結合法**の技法が開発されている(Madnick, 1973)(すなわち, 最悪の場合の限界が $\frac{M}{N}+1$ から2に減じた). この技法は, LRU や FIFO などの従

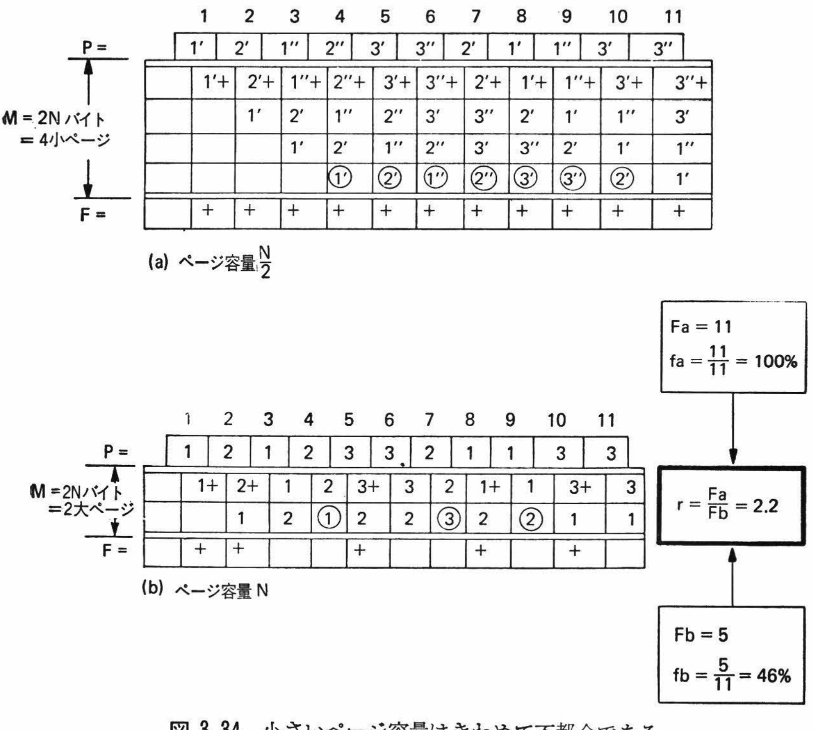


図 3.34 小さいページ容量はきわめて不都合である

来の技法にある種の制約を課すものである．

項結合の基となっている基本概念は単純である．第1，元の大きいページPの2つの部分P'とP''は同一視できること（すなわち，ページの集合は2項子の集合とみなされること）．第2，対応する大きなページPがMから置換えられない間はページP'やP''は置換えられることがないように置換えの順序づけを2項子の両方の要素に適用すること（すなわち，2項子は順序づけに関して結合している）．この手法の組込みは，LRUやFIFOなどの使用する置換えアルゴリズムによって多少変化することがある．項結合法を具体化できるような置換えアルゴリズムは“**項の結合可**”と呼ばれる．

図3.35は，図3.33のLRUの例に対する項結合法の応用を説明している．LRU置換えが行なわれたときには$r=2.2$であったのが，例の場合には$r=2$に制限されていることに注意してほしい．このLRUの例で用いられた項結合法では，

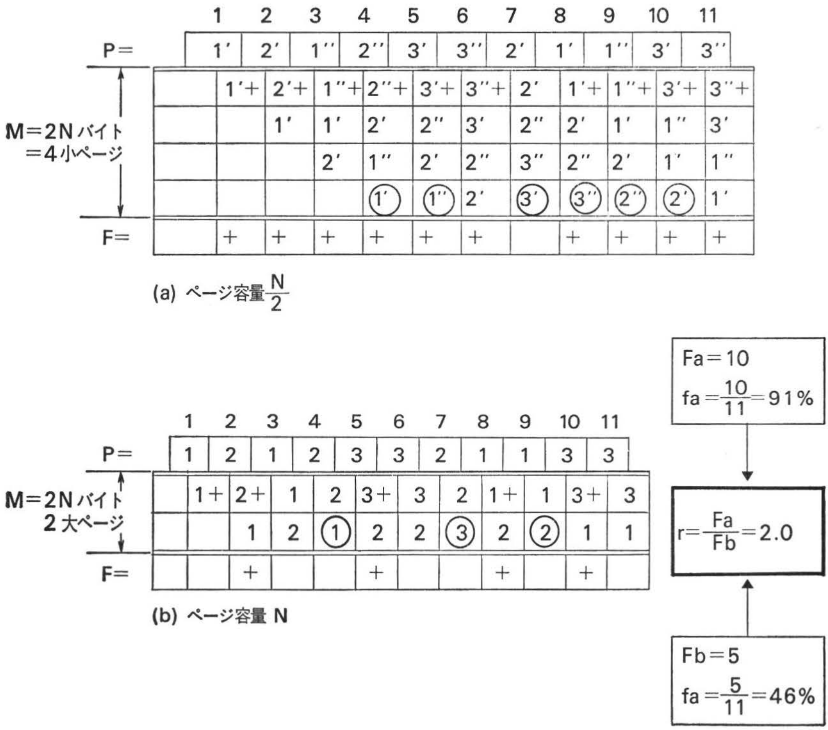


図3.35 項結合を行なうLRU置換えアルゴリズム

1. 一方が参照されるか，または
2. 他方がすでに M に入っている

ならば，P' と P'' を表の先頭に移す．

3.6.3 効率とプログラム動作の考察

デマンド・ページングを行なうと，同時により多くのジョブを多重プログラミングすることが可能となり（すなわち，物理的な記憶域の大きさの制限が除かれる），システム効率がよくなる．仮想記憶は“仮想”とはいっても完全に“無料”なのではない．ページ割込みの処理を行なうと，オペレーティング・システムのオーバヘッドによる処理装置使用量の増大と，ページ入出力による入出力の増加によって“経費”がかかる．これらの妥協点を求めるにはプログラムの参照動作の性質を考察する必要がある．

3.6.3.1 番地参照の型

図 3.36 は実際の IBM 360 プログラムの番地参照の型を示したものである（Hatfield，1972 による）．この図を理解するには記憶装置の各バイトに小さな電球が接続されていると考えるとよい．この電球はバイトがデータもしくは命令として参照されたとき一瞬点灯する．長い列になって並べられている電球にカメラが向けられているとしよう．カメラのシヤッターが開き，プログラムが一瞬走り，ランプが点滅する．シヤッターが閉じフィルムが送られる．これをプログラムが完了するまで繰返す．図 3.36 の垂直の列は短い時間の間に参照された番地を示している．垂直軸は 4K ページの開始点を目盛ってある．たとえば，ページ 25 から 28 はプログラム実行の終りの方にならないと参照されないことに注意してほしい．この図は，実際には前節で述べた番地の軌跡を決定するためにプログラム実行をシミュレートして作成されたものである．

プログラムの参照の型は，図 3.36 が示しているように，ランダムではないことに注意してほしい．実行点の異なった段階（たとえば，入力，表の設定，計算，表の探索など）に対して一定の型が対応している．プログラムの動作は，図では傾斜線となるような“大きな表の直線的な処理”といったように，認識することができる．

ページングの効率をよくするためには，プログラマは参照をできるだけ“かためる”，すなわち“局所化”することが重要である．たとえば，大きな行列 $A(n,\ m)$ が $A(1,\ 1)$，$A(1,\ 2)$，…，$A(1,\ m)$，$A(2,\ 1)$，$A(2,\ 2)$，…，

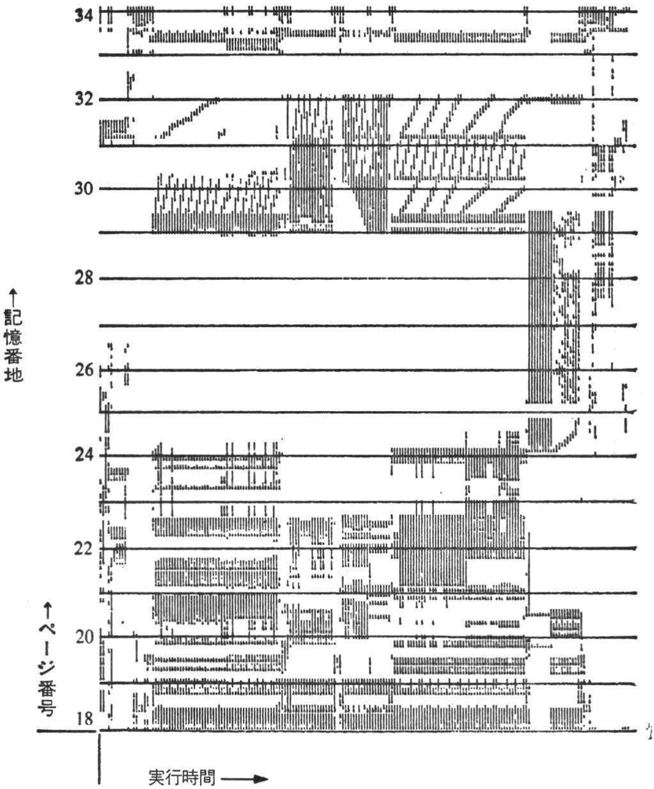


図 3.36 記憶参照のパターンの例（Hatfield；1972 より引用）

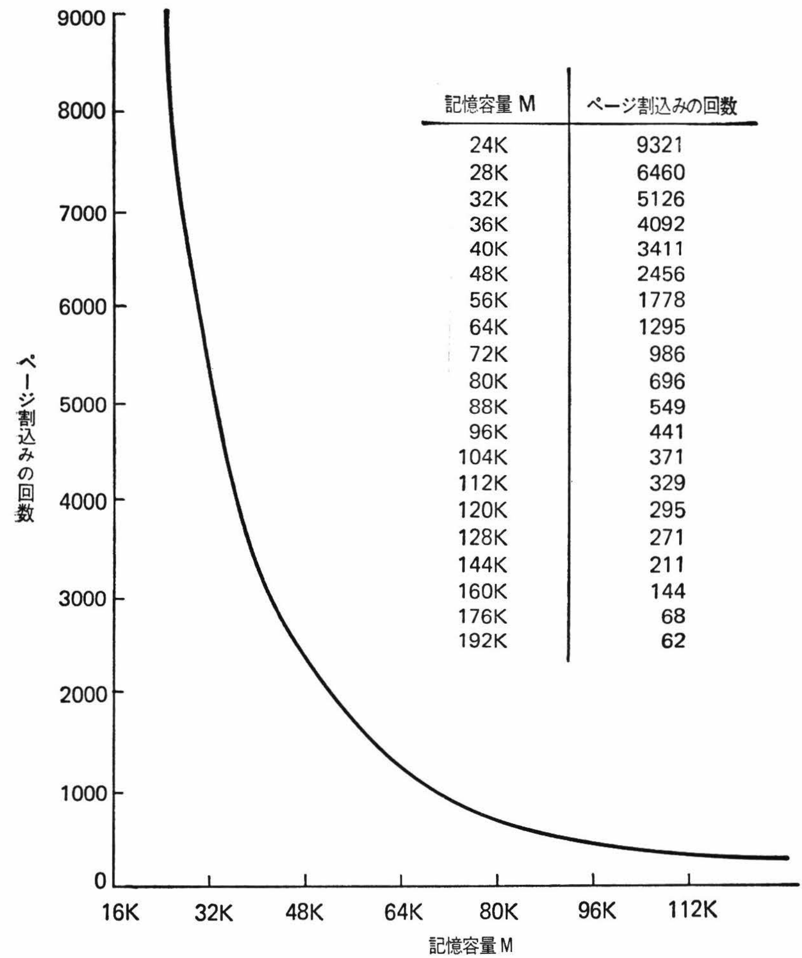
A(2, m)，…の順で並んでいるとしたら，行列処理の順を A (1, 1)，A(2, 1)，A(3, 1) とするより，可能ならば A(1, 1)，A(1, 2)，A(1, 3) とする方が優れている．デマンド・ページングのシステムの効率のよいプログラミングについては多数の提案が行なわれている（Hatfield, 1971；Morrison, 1973）．

3.6.3.2 主記憶容量とページ割込み回数の関係

プログラムで発生するページ割込みの回数はページを格納するために使用できる物理的な記憶域の容量の関数である．図 3.37 は，IBM 360 での実際のプログラムに対する関係を示すが，この形状は典型的なものである．

プログラムが小容量の記憶装置に制限されているときには，ページ割込みが多く起こることに注意してほしい．記憶容量が 32K から 40K に 8K 増加するとページ割込みは 1715回減少する．64K から 72K に同じく 8K 増加したときには，ページ割込みは 309 回しか減少しない．

たいていのプログラムでは，ある値以上では記憶容量を増加してもさほど大きな改善とはならないが，この値より小しでも容量を減少させると顕著にペー



記憶容量 M	ページ割込みの回数
24K	9321
28K	6460
32K	5126
36K	4092
40K	3411
48K	2456
56K	1778
64K	1295
72K	986
80K	696
88K	549
96K	441
104K	371
112K	329
120K	295
128K	271
144K	211
160K	144
176K	68
192K	62

図 3.37 記憶容量とページ割込み回数の関係の例

ジ割込み回数が増すような閾値が存在する．この値はプログラムによって大きく変化する．1つの研究によると物理的な記憶容量の閾値は，番地空間（仮想記憶）の全容量の半分位のことが多い（Coffman, 1968）．図 3.37 では番地空間の全容量は 256K バイトである．この閾値はしばしばプログラムの**“ワーキング・セット容量”**と呼ばれる．これは，プログラムが効率よく働くためには主記憶中に最小限のページが必要であることを意味している．（これはワーキング・セットの正式な定義ではない．この定義は次節にて行なわれる.）

3.6.3.3 ワーキング・セットの理論

記憶容量とページ割込み回数の関係はプログラムのページング動作の総合的な尺度を示す（Bard, 1973）．プログラムの動作は，図 3.36 に示すように，実行中つねに一様ではない．さらに，多重プログラミングを行なっている場合には複数のジョブが物理的な記憶領域を競合しているので，1つのジョブの使用している記憶ブロックの数は一定であるとは限らない．

より局所的な効率の尺度は，一定の時間内でプログラムが参照するページ数である**ワーキング・セットの逐次的な増加量**である（Denning, 1968）．時刻 $t-\Delta t$ と t の間に参照されるページの一覧表 $W(t, \Delta t)$ は**ワーキング・セット**と呼ばれる．ワーキング・セット容量 $\omega(t, \Delta t)$ は $W(t, \Delta t)$ に含まれるページ数である．ワーキング・セット容量は Δt の単調増加関数である．すなわち，時間が長いほど参照されるページ数は大きく，極限値は実際必要となるページ数の合計 L である．L は番地空間の全容量より大であることはない．

Δt を 10 ミリ秒あるいは 10,000 命令（実際の経過時間よりもむしろ実行した命令数で計算されることが多い）程度の時間に選ぶと，プログラムの現在の所要記憶容量――ワーキング・セット容量――を推定することが可能となる．このようにするとオペレーティング・システムは，どのジョブにもっと記憶ブロックを割付け，どのジョブの記憶域をあけ渡したらよいかを決定することができる．

プログラムのワーキング・セット容量の時間変化を調整し，記憶域を要求し過ぎてページング動作が過度にならないようにするためには，記憶管理と処理装置管理の動作を調整しなければならない．もしジョブのワーキング・セットを収容するのに十分な記憶域がなければ，ジョブに処理装置を割付けることは無意味である．第8章ではこの現象の解析を行なう．

3.6.3.4 局所性

仮想記憶システムの効率的な動作は，プログラムにおける参照の局所性の程度に依存する．局所性は，時間的な局所性と空間的な局所性に分けられる（Madnick, 1973）.

時間的な局所性は，たいていのプログラムで観測される性質で，一度その場所——データまたは命令——が参照されると，すぐにまた参照されることが多いことを意味する．

空間的な局所性は，一度ある場所が参照されると近接の場所はその直後に参照される可能性があることを意味する．このような動作は，命令実行が逐次的であることや直線的なデータ構造（たとえば，配列），およびプログラマは共通に使用する変数をまとめて置く傾向があるなどのプログラムの構造によって裏づけることができる．

局所性の度合いはプログラムによって異なっている．一般に，局所性の高いプログラムを開発することが望ましい．これに適するプログラミング技法はMorrison, 1973 および Brown, 1968 が論じている．

3.6.4 利点

デマンド・ページングによる記憶管理は，ページングによる記憶管理で先に述べた利点のすべてを有している．特に，断片化は解消され，詰め合せは不必要である．

さらに幾つかの利点が存在する：

1. **大容量の仮想記憶**　ジョブの番地空間の容量は物理的な記憶容量に制限されない．これによって大規模な計算機と小規模な計算機の互換性がよくなる（たとえば，普通1024Kの計算機で実行していた512Kバイトのジョブは，必要ならば，256Kの計算機でも実行できる）．
2. **もっと効率のよい記憶装置の利用**　使用されなかったり，滅多に使用されない番地空間の部分は物理的な記憶域に収容されない．ジョブの番地空間の25%以上もの部分は，特殊な条件の下（たとえば，誤り処理ルーチン）でしか使用されないので，ロードする必要がない．
3. **制限のない多重プログラミング**　前述の方式では，多重プログラミングの多重度は，通常，物理的な記憶容量で制限された．すなわち，ジョブの番地空間の合計は物理的な記憶容量より大であってはならなかっ

た．デマンド・ページングではこのような制限はない．必ずしも望ましいことではないが，わずか256Kバイトの記憶容量の計算機で50とか100ジョブを多重プログラミングすることも可能である．

3.6.5 欠　点

第3.5.4節で指摘したページングによる記憶管理の欠点（ハードウェアの経費，ページ余り，処理装置のオーバヘッド，および表のための記憶域）に加えて，デマンド・ページングの記憶管理の場合には，新しい問題点が生じる：

1. 表の個数と処理装置のページ割込み処理のためのオーバヘッドは，単純ページングによる記憶管理技法の場合より大きい．
2. ジョブの番地空間の容量や多重プログラミングの多重度にはっきりした限界がないので，スラッシング（利用者のジョブがわずかしか進まないのに，ページ割込みの処理のために，システムで過度に処理装置のオーバヘッドが費やされること）を防止する手法を開発する必要がある．極端なスラッシングの場合には，99%以上もの処理装置時間がオーバヘッドの動作に費やされ，利用者ジョブには1%以下しか使用されない．（スラッシング防止のための技法については第8章で述べる．）

3.7 セグメンテーションによる記憶管理

前述した記憶管理の手法では，物理的な記憶装置は，分割，再配置，ページングなどの種々の方法で取扱われた．しかし，これらの実行は利用者プログラムからは見えないものであった．1つの直接的で連続な番地空間がつねに与えられていた．効率がよく，プログラマにとって便利な番地空間を与える方法がほかに存在するであろうか．存在する．それはセグメンテーションである．

セグメントとは，サブルーチンとか配列とかデータ領域とかいった情報の論理的な単位であると定義される．したがって，各ジョブの番地空間は実際には図3.38に示したようなセグメントの集合で構成される．**セグメンテーション**はこれらのセグメントを管理する技法である．

セグメンテーションを行なう場合には，番地空間の参照には次の2つの成分が必要である：

1. セグメントの指定
2. セグメント内の番地指定

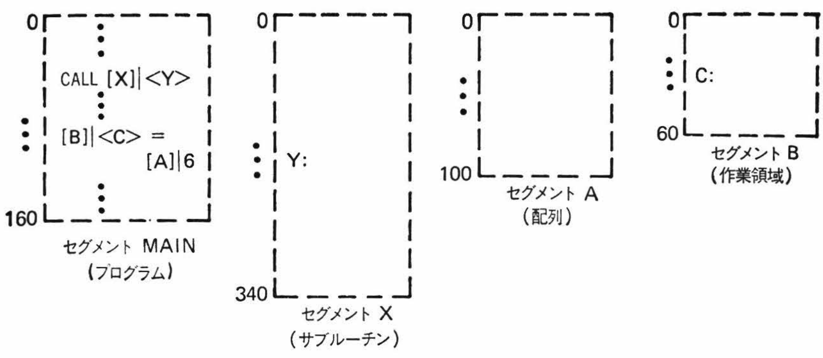


図 3.38 セグメンテーションによる番地空間

たとえば，参照は次のように行なわれる：

CALL	[X]|〈Y〉	サブルーチンXの中の入口Yに制御を移す
LOAD	1,[A]|6	配列Aの6番地をレジスタ1にロードする
STORE	1,[B]|〈C〉	レジスタ1をセグメントBの中の番地Cに格納する

従来の番地空間とは異なって，セグメンテーションを行なう番地空間はプログラマに“見える”けれども，この違いは必ずしもプログラミングに影響しない．たいていのプログラミング言語では次のような文がある：

CALL Y；
C=A(6)；

単一で直線的な番地空間では，コンパイラやローダはセグメントの参照を単一の番地に変換する必要があった．たとえば，YがサブルーチンXの中の 120 番地に置かれた入口で，X が 3460 番地にロードされたときには，CALL Y は CALL 3580 となる．同様にして， A が 3800 番地に置かれたバイトの配列ならば，A(6) は 3806 番地を参照する．元の構造はこの変換で失われていることに注意してほしい．もし 3803 番地で誤りが発生したとしても，3803 が実は A(3) であるという表示は陽には存在しない．

普通，セグメンテーションを行なう番地空間においては，各セグメントは，[X]=3,［A］=5というように，番号がつけられている．したがって，CALL Y は CALL [3]|120となり，A(6) は [5]|6となって，セグメントとセグメント内で参照される場所とを陽に示している．

セグメンテーションを行なう番地空間は，ページングによる記憶管理で使用されたのと同様な番地写像のハードウェアを用いて組込むことができる．図 3.39 は，各セグメントの主記憶中での番地を示すセグメント・マップ表（SMT）を示している．2次元のセグメンテーションの番地から直線的な物理的記憶番地への変換は，コンパイラやローダによって静的に行なわれるのではなく，実行中に自動的に行なわれる．

ページとセグメントの番地写像のハードウェアは類似しているので，ページングとセグメンテーションの区別がしばしば混乱することがある．本節では，概念的な相違点を強調しておこう．主たる相違点は次のとおりである．セグメントは情報の“論理的な”単位であり，利用者プログラムの中でこれを見ることができ，容量は任意である．ページは情報の“物理的な”単位であり，記憶管理のためだけに使用されて利用者プログラムにはその存在がわからず，固定の容量，たとえば 4K など，である．前のことを思い出すと，セグメントは，多重区画割付けの議論（第 3.3.2.6 節）で導入されたジョブの“部分”という概念をもっと正確に表わすものである．

セグメンテーションを行なう記憶管理技法には幾つかの利点がある(Dennis, 1965, 1966；Daley, 1969；Randell, 1967；Denning, 1970)：

1. **断片化の解消**　セグメントを移動させることにより，断片となった記憶域を1つの空き領域に統合することができる．
2. **仮想記憶を作る**　実際に使用中のセグメントだけを主記憶に格納することによって，ジョブの番地空間の容量の合計を物理的な記憶容量以上とすることができる．
3. **動的にセグメントの長さを延長することができる．あるいは自動的な限界検査が可能**　実行中にセグメント容量を増大しなければならない場合（たとえば，表の領域が一杯となった），“領域外”への参照は，セグメント・マップ表の各欄に記録されている容量によって検出することができる．これを検出すると，この条件に対するオペレーティング・システムの処理方式に従って，セグメント容量を増大させる場合には動的な領域割付けが，誤りの場合には誤りの表示が行なわれる．

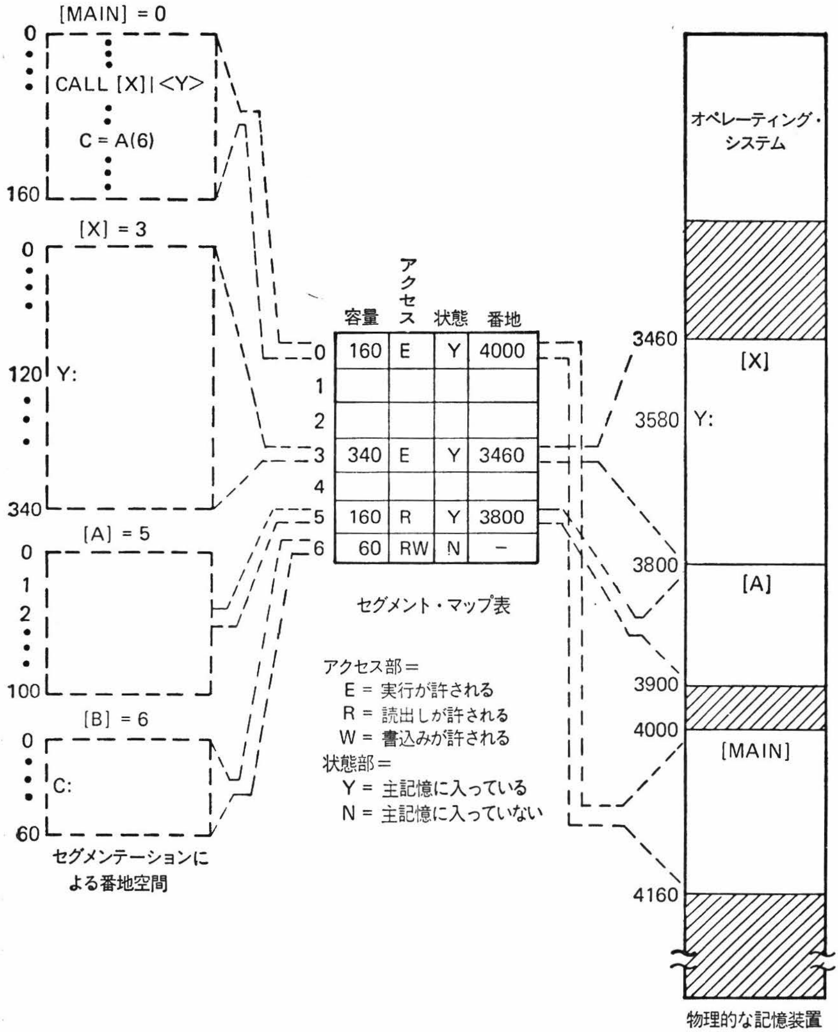


図 3.39 セグメンテーションによる記憶域の写像

4. ダイナミック・リンキングとローディング　たいていのシステムでは，サブルーチンやデータ領域は，実行の開始に先立って，結合やローディング処理の段階でジョブの番地空間に領域を割付けられ結合されている (Donovan, 1972 ; Presser, 1972). 多数のサブルーチンを使用する

大きなプログラムの場合には，この結合処理は複雑で時間がかかり，時として実際の実行よりも時間がかかることもある．しかも，結合されたサブルーチンの多くは，特定の実行の間には実際に呼出されないことも多い．したがって，セグメントが参照されるまで，結合を行なわないことにすれば，不必要な結合を省くことができる．

5. **セグメントの共有を容易にする**（データ領域と手続き）　2つのジョブが平方根のルーチンを使用するとき，主記憶中に別個の2つの写しが存在するのは無駄で，好ましいことではない．

6. **制御されたアクセスの実施**　各セグメントへの参照は制御されなければならない．目次は読出し専用に限定されるべきであるが，作業用領域は読出しと同時に書込みも行なわれる．純手続きセグメントは，事故で破壊してしまう危険があるので書込みが認められないし，手続きを“盗み”出されてはいけないので読出しも認められない．参照は実行だけに限定されるべきである．セグメントを共用する場合には，読出し，書込み，および実行の権利はセグメントとセグメントの利用者との関係によって割付けられる．

セグメンテーションによる記憶管理は，1960年代の初期以来，主として Burroughs B5000 およびその後に続く計算機などで使用されている．もっと最近になって，セグメンテーションは Honeywell 6180 MULTICS システム，および幾分小規模ながら IBM 370 の OS/VS2 システムでも採用されている．

セグメンテーションによる記憶管理の4つの機能は，再配置型分割割付けとデマンド・ページングによる記憶管理を混合したものとなっている：

1. 状態の把握は4つの表を用いて行なわれる．
   a．セグメント・マップ表——番地空間に1つ
   b．空き領域表——システムに1つ
   c．セグメント名表——番地空間に1つ
   d．アクティヴ・セグメントの状態表——システムに1つ
2. どのプロセスにいつ記憶域を与えるかという方針は，仮想記憶の使用が制限されていれば（すなわち，仮想記憶域の容量が物理的な記憶域の容量と同程度ならば），ジョブ・スケジューラによって静的に決定される．一般的な仮想記憶の場合には，記憶域に関する要求はセグメントを“要求”する割込みに基づいて動的に決定される．
3. 割付け——セグメントを割付けるには，十分な容量の空き領域がなけ

ればならない．必要ならば，詰め合せが行なわれる．

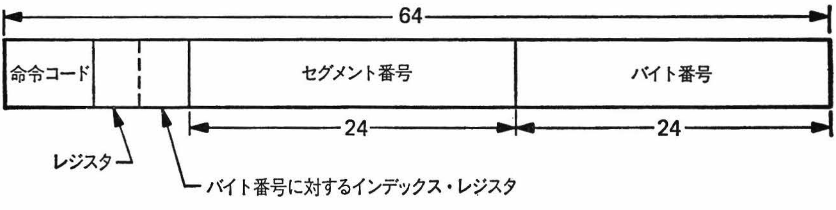
4. 解 放——セグメントを割付けるために十分な容量の空き領域が求まらないときには，現在割付けられているセグメントを1つ以上解放しなければならない．解放された領域は空きとなる．ジョブが終了したとき，ジョブのセグメントが使用していた記憶域は空きとなる．

3.7.1 ハードウェア

セグメンテーションの記憶管理を組込むハードウェアには多数の変形が存在する．本節では重要なハードウェアの概念を述べる．詳細について，あるいは別の方法については参考文献を参照してほしい (Dennis, 1966, 1967 ; Daley, 1969 ; Denning, 1970).

3.7.1.1 セグメンテーションの番地の形成

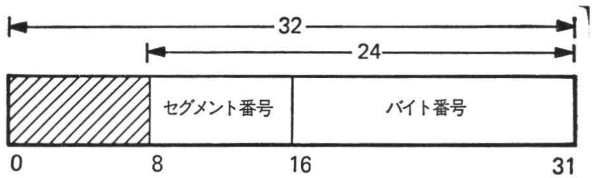
セグメンテーションを行なう場合，セグメント番号とバイト番号の2つの成分を持つ番地を生成しなければならない．これを行なう1つの方法は，各命令に次図のように2つの番地部を設けることである：



このようにすると LOAD 1,5|8 というような命令を容易に形成することができる．ここで5はセグメント番号，8はセグメント内のバイト番号である．このような命令の長さを短縮し，同時にもっと柔軟性を持たせるために，セグメント・ベース・レジスタおよびバイト・ベース・レジスタを，命令の中に置かれたセグメント・オフセットあるいはバイト・オフセットと共に使用することができる．このような番地指定法は Burroughs 5500 や Honeywell 6180 で採用されている．

別の方法は，線型の番地形成技法によって，実効番地を生成し，求まった実効番地のある部分をセグメント番号とし，残りの部分をバイト番号とする方法である．たとえば，IBM 370 では 24 ビットの実効番地を次図のように8ビッ

トのセグメント番号と16ビットのバイト番号として解釈することができ，最大64Kバイトのセグメントを256個まで番地指定できる：



IBM 360 モデル 67（タイムシェアリングのために設計された IBM 360シリーズの特殊なモデル）で行なわれているように，32ビットの実効番地を使用するとより多くのより大きいセグメント——最大4096セグメント，最大100万バイト——の指定が可能となる．これに対して，Honeywell 6180 では最大 256K 個のセグメント，最大100万バイトまで可能である．

後者の方法は，命令の中では単一の番地部を使用しているので，セグメンテーションの番地空間が利用者にわからないようにすることができるという利点（見方によっては欠点でもある）を有している．単一の番地形成を行なうために，1つのセグメントの最終バイトは（最大容量の場合には），次のセグメントの第1バイトと論理的に隣り合うことになる．すなわち，隣り合った4つの64K セグメントは1つの256K セグメントとして取扱うこともできる．命令の中にセグメントとバイトの番地部が別々に設けられているときにはこのようなことは不可能である．

3.7.1.2 番地の写像

必要となるセグメント番地の写像のハードウェアは，第3.6.1節で述べたページ写像のハードウェアと類似したものである．しかし，セグメントとページの相違のために幾つかの点が異なる．セグメントの長さは任意であるので，参照するバイトの番地がセグメントの領域に入っているか検査する必要がある．

セグメントのレベルでの保護は，セグメントがプログラマにとって論理的に意味のある実体であることから，ページや記憶ブロックの保護よりももっと必要とされる．したがって，番地写像の機構は参照が許されているかどうかを検査しなければならない．

セグメントは主記憶から取除かれ，2次記憶に置かれることもあるので，セグメントが現在主記憶に入っているかどうかを表示することが必要である．主記憶に入っていない場合には，割込みを発生してオペレーティング・システム

に通知しなければならない．オペレーティング・システムが，どのセグメントを主記憶中に残し，どれを取除くかを決定するのを助けるために，セグメント・マップ表の各欄に“参照ビット”および“変更ビット”が設けられることがある．

図 3.39 は一般的なセグメント・マップ表の様式を，図 3.40 は番地写像の処理で起こる一連の動作を示している．個々の組込みにおいては，アクセス保護や参照/変更の情報などの機能が含まれないこともあるし，Honeywell 6180 の“リング保護機構”などの機能が追加されることもある．

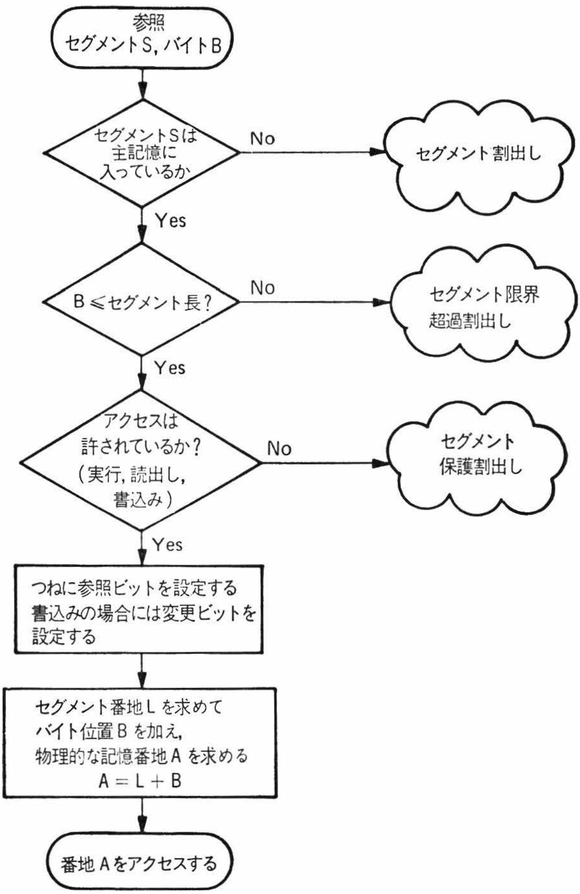


図 3.40 セグメントの番地写像

3.7.2 ソフトウェアのアルゴリズム

オペレーティング・システムにおいて，セグメンテーションのために必要となるソフトウェアのアルゴリズムの多くは，デマンド・ページングのそれと類似している．本節では，セグメンテーションによる記憶管理の6つの利点がいかにして達成されるかを説明する．他の記憶管理技法を用いても，必ずしも全部ではないが，前述の利点の多くのものは達成することができる．しかし，一般にはセグメンテーションによると組込みがはるかに単純である．

3.7.2.1 断片化の解消

セグメント・マップ表は多数の再配置／限界レジスタとして作用する．1つの欄の番地部を再調整すると，ジョブには見かけ上の変化なしに，セグメントを主記憶の中で移動させることができる．図 3.39 では，セグメント X, A および MAIN を移動させて，3つの空き領域を結合して1つの大きな領域にしている．

3.7.2.2 仮想記憶を構成する

仮想記憶はデマンド・ページングの場合と同様にして構成される．すべてのセグメントの写しは2次記憶に収容される．主記憶に入っていないセグメントが参照されると“セグメントなし割込み”が発生する．オペレーティング・システムは，記憶域を詰め合せたりセグメントを取除いたりして，必要となったセグメントのために領域を確保し，次いでそのセグメントを主記憶にロードしなければならない．

セグメンテーションによって仮想記憶を構成するのとデマンド・ページングによって行なうことの利害得失に関して多くの議論が行なわれている．セグメンテーションに関する主たる利点の1つとしては，情報として意味のある1単位のセグメントが交換されることである．たとえば，400バイトのセグメントの中が参照されたとすると，固定長の4Kバイトのページではなく，400バイトだけを主記憶にロードすればよい．16Kバイトの容量の配列が必要となれば，別々に4回ページ割込みを起こすのではなく，セグメント全体を1回でロードする．この反面，可変長のセグメントを管理するのは仮想記憶の主要な問題点であろう．任意長のセグメントを2次記憶に格納することは，しばしば複雑な領域割付け，無駄な領域，あるいはセグメント容量の厳しい制限（たとえば，1Kバイトの倍数でなければならないなど）の点で問題が生じる．また，

このようなセグメントを主記憶中で操作すると，詰め合せによる余分のオーバヘッドが生じることもある．セグメント全体をロードしなければならないということには幾つかの重大な問題点がある．512K バイトのセグメントの中で1バイトだけが必要であっても，セグメント全体をロードしなければならない．もし 200,000 語の配列，すなわち 800K バイトのセグメントが必要となり，主記憶が 512K バイトであればもっと困ったことになる．1つのセグメントの大きさは物理的な記憶容量より大であってはならない．

第 3.8 節で論じるセグメンテーションとページングを併用した技法によって，上述の問題点のほとんどは解消する．ページングが行なわれない場合には，ある種の規約を設けてこれらの問題を最小にすることができる．たとえば，セグメントはやや小さ目の容量とし，基本容量（たとえば 1K バイト）の倍数として格納し，セグメントの入れ換えの回数を少なくするためにセグメント容量を物理的な記憶容量と比べてはるかに大きくしてはならない．

3.7.2.3 動的なセグメントの拡張を可能にする

ある種のデータ・セグメントの長さは入力データに依存する．たとえば，アセンブラの記号表の大きさは入力された原始プログラムの中で使用される記号数によって定まる．このような場合，動的に拡張するセグメントを作ることができる．すなわち，プログラマは，セグメントは任意長であり，新しい欄をセグメントに追加して行くだけでよいと考えることができる．実際に，セグメント長は非常に小さい値から始まる．プログラマが新しい欄をセグメントに格納しようとすると“限界外”割込みが発生する．次いで，オペレーティング・システムに制御が移されるので，セグメント長を増大させる．このためには，詰め合せや，場合によっては他のセグメントを主記憶から取除かねばならないこともある．セグメントを大きい領域へ移動させた後に，セグメント・マップ表の中のセグメント番地と長さの部分が調整される．

多くのデータ・セグメントは固定容量である．この場合には，限界外割込みは誤りの状態であることを表わしている．この処理はセグメントの属性によって決定される．追加属性は自動的な拡張を意図するものとみなされる．

3.7.2.4 ダイナミック・リンキングとローディング

大規模で複雑なプログラムは多数のサブルーチンから構成されている．これらのサブルーチンを全部まとめて結合し，一時にすべてを主記憶にロードすることは，特定の実行時にこれらの多くが使用される見込みがない場合には，特

に，不経済なことである．**ダイナミック・リンキングおよびローディング**を行なっている場合には，最初メイン・プログラムだけがロードされる．メイン・プログラムで他の手続きが参照されると，そのセグメントがロードされ，番地の結合が参照時に行なわれる．この方法は，サブルーチンの結合が実際に参照が行なわれるまで遅延されることから，しばしば**遅延結合**とも呼ばれる．

ダイナミック・リンキングを行なうには幾とおりかの方法がある．この組込みのためにはセグメンテーションを行なわねばならないということはないが，セグメンテーションが行なわれていると具合がよい．MULTICS システムで採用している技法では，間接番地指定とリンケージ割出し表示子の 2 つの付加的なハードウェアが必要である．

100 番地に数 320 が，320 番地に数 400 が格納されているとする．**直接番地指定**の命令 LOAD 100 は 100 番地の内容 (320) をロードすることを意味する．

間接番地指定で LOAD* 100 とすると，データ・オペランドの内容を番地指定のための番地として使用する．LOAD* 100 は LOAD 320 とみなされ，320 番地の内容 (400) がロードされる．図 3.41 は直接番地指定と間接番地指定の相違を示している．

間接番地指定を行なうとき，番地を格納している語を**間接番地語**という．この語には，番地のほかに状態ビットを格納することもできる．たとえば，間接

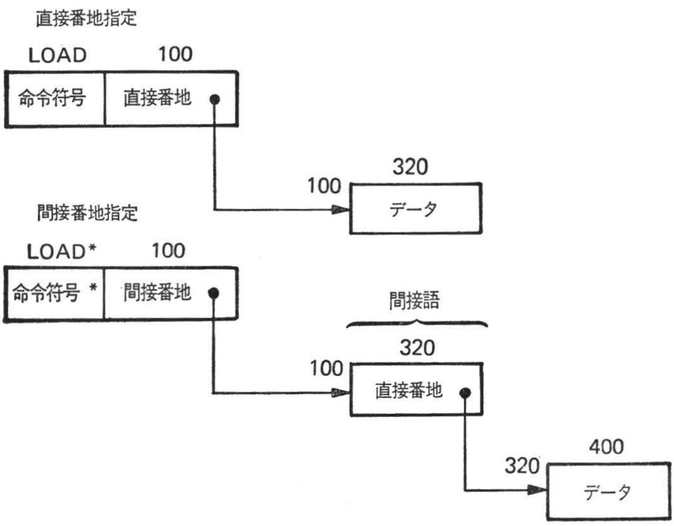


図 3.41　直接番地指定と間接番地指定

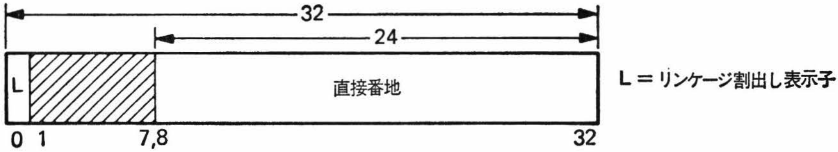


図 3.42 間接番地語の書式

番地語の書式は 図 3.42のようになる.

もしリンケージ割出し表示子が1に設定されていて，この語が間接番地語として使用されると**リンケージ割出し**が発生する．この表示子が0ならば間接番地指定は先に論じたように進行する.

間接番地指定とリンケージ割出しの2つの機構を用い，図 3.43 に示したようにしてダイナミック・リンキングの機能を組込むことができる．プログラムの中で他のセグメントを LOAD [X]|⟨Y⟩ のように記号で参照しているとき，コンパイラやアセンブラは間接番地指定の命令を生成する（図 3.43a の60番地を参照．間接番地語はリンケージ割出し表示子が設定されており，番地は記号参照（たとえば，7“[X]|⟨Y⟩”）を表わす文字列の番地を示している).

LOAD [X]|⟨Y⟩ 命令が実行されるとリンケージ割出しが発生する．制御はオペレーティング・システムに移されて，その原因がリンケージ割出しであることが判明し，割込み情報によって 3|108 番地の記号名が見つけられる．要求されたセグメント［X］は，情報管理（第6章）の援助によってロードしなければならない．このセグメントにセグメント番号が割付けられ（この場合は4），間接番地語は［X]|⟨Y⟩ に対するセグメント|バイト番地に変更される．結合後の状態は図3.43bに示されている．割込まれた3|60番地の命令はPSWをロードすることによって再開することができる．今回は必要なデータ015571が正しくロードされる．これ以後は，セグメント [MAIN] の中でこの命令が実行されてもリンケージ割出しが起こらずに動作する.

ダイナミック・リンキングによって引き起こされるオーバヘッドを軽減するのに幾つかの技法が使用できる．たとえば，同じ記号番地を参照するものは同一の間接番地語を使用することができる．図 3.43 で STORE 1, [X]|⟨Y⟩ 命令があると，この命令はSTORE* 1, 3|100 に変更される．3|100 番地にはすべての［X]|⟨Y⟩ への参照を結合するために使用する間接番地語が格納されている．したがって，この参照が手続きの中で最初に行なわれたとき結合が行なわれ，この記号を参照するすべての場合に使用される.

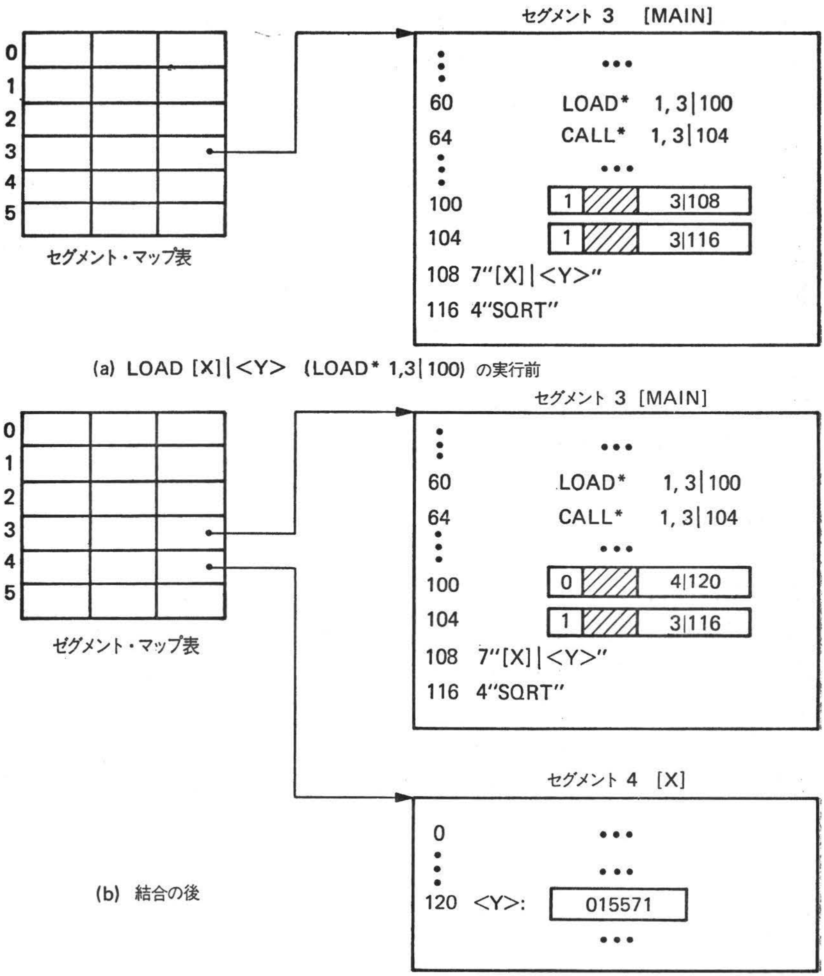


図 3.43 ダイナミック・リンキング

手続きはそれぞれ別々にコンパイルされることから，これらのリンケージのための間接番地語は各手続きにとって局所的である．一度記号参照が行なわれると，以後同様の手順を省き変換の一貫性を保つ目的で，参照記号の複写と割付けられた番地はアクティヴ参照表（ART）——文献では**アクティヴ名前表**や**既知セグメント表**とも呼ばれている——に保存される．したがって，リンケー

ジ割出しが発生すると，情報管理に依頼したり手間のかかる結合処理を行なう前に，アクティヴ参照表を調べる（その記号によるセグメント参照がすでに別の手続きで行なわれていて，セグメント番号を割付けられているかもしれないと期待しながら）．もしすでにアクティヴ参照表に入っていれば，リンケージのための間接番地語は単に表から複写される．

ダイナミック・リンキングについてはこのほかに幾つかの問題点がある．1つの問題点は，結合処理によってリンケージのための間接番地語が変更されることである．実行中に手続きの変更を認められない**純手続き**の概念とセグメントの実行中の変更とは矛盾する．ダイナミック・リンキングを行なう場合で，手続きの純枠性を保持する方法の1つは，間接番地語やある種の一時的な変数，内部変数などを含めて，すべての変更を受ける情報を**リンケージ・セグメント**と呼ばれる別個のセグメントに置くことである．したがって，各原始プログラムに対して，コンパイラは少なくとも純粋なものと純粋でないものとの2つの別個のセグメントを生成する必要がある．この要件は，IBM 360 の PL/I コンパイラと Honeywell MULTICS の PL/I コンパイラで満たされている．

これらの2つのセグメントの使用法と相互の関係に関しては多数の組込み方法がある．図 3.44 は MULTICS の機構を簡単に示している (Dennis, 1965；Daley, 1968)．実行中にその手続きセグメントとリンケージ・セグメントを番地指定するために2つのベース・レジスタ（手続き用レジスタ，リンケージ・レジスタ）が使用されている．間接番地指定の命令もリンケージのための間接番地指定語もこれらのベース・レジスタを使用している．図 3.44 は，コンパイラで作成された，ダイナミック・リンキングが行なわれる前の，手続きセグ

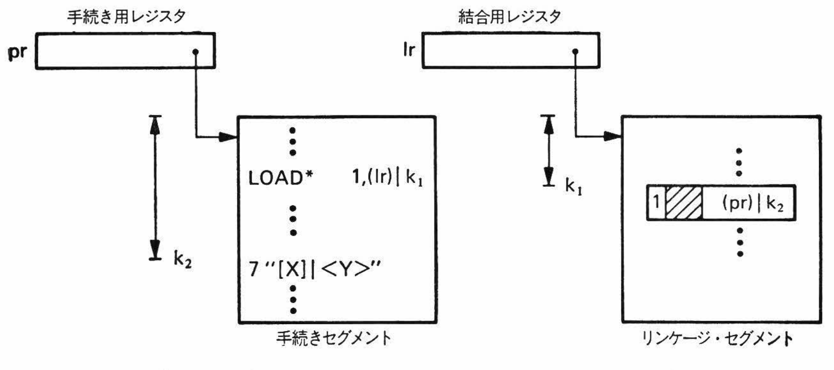


図 3.44 手続きセグメントとリンケージ・セグメントの分離

メントとリンケージ・セグメントとの形を示している．定数 K1 および K2 はコンパイラによって決定される．リンケージの処理は前述したように行なわれるが，手続きセグメントの純粋性は保たれている．

セグメント間の制御移行（すなわち，サブルーチンの呼出し）はさらに問題を複雑にする．レジスタの値や引数，復帰番地は，通常，サブルーチン結合のためのスタック・セグメントの中に退避される．スタック・レジスタは現在のスタック領域を指している．1つの手続きセグメントから別の手続きセグメントに制御を移すためには，手続きベース・レジスタ，リンケージ・ベース・レジスタ，およびスタック・ベース・レジスタを正しく変更しなければならない．MULTICS システムで採用されている1つの方法は Daley (1968) および Organick (1972) によって論じられている．

3.7.2.5 セグメントの共用

前述した記憶管理技法では，2つのジョブが同じ手続き——たとえば，平方根のサブルーチン——を利用する場合，各ジョブの番地空間に別個にサブルーチンの写しが置かれていた．このようなことは，共通のサブルーチン・ライブラリ，システム・ユーティリティ（たとえば，コンパイラ，アセンブラなど），および標準の応用プログラムなどの手続の場合によく起こる．このような場合には記憶域の大部分は手続きの写しで詰まってしまう．ダイナミック・リンキングとセグメンテーションが行なわれるとこのような複写の作成を解消することができる．

図 3.45 は，手続き COS の複写を持っている2つのジョブの番地空間を示している．セグメント番号は，ダイナミック・リンキングの処理で動的に割当てられている．各ジョブにおいてどの手続きが使用されるかを前もって予言することは不可能であるから，共通的な手続きには必ずしも同じセグメント番地が割当てられるとは限らない．たとえば，COS はジョブ1の番地空間ではセグメント番地が6，ジョブ2ではセグメント番号が3である．セグメントの共用は 図 3.46 に示されている．図では，COS に対する2つのジョブのセグメント・マップ表の欄が物理的に同一の COS サブルーチンを指している．（図では，リンケージ・セグメントやスタック・セグメントが示されていないが，これらは別に用意されているものとする．）

多重プログラミングを行なっている場合にセグメントの共用を効果的に行なうには，手続きは**再入可能（リエントラント）**でなければならない．純手続きは，レジスタに格納された変数か，ジョブに関連した別個のデータ・セグメン

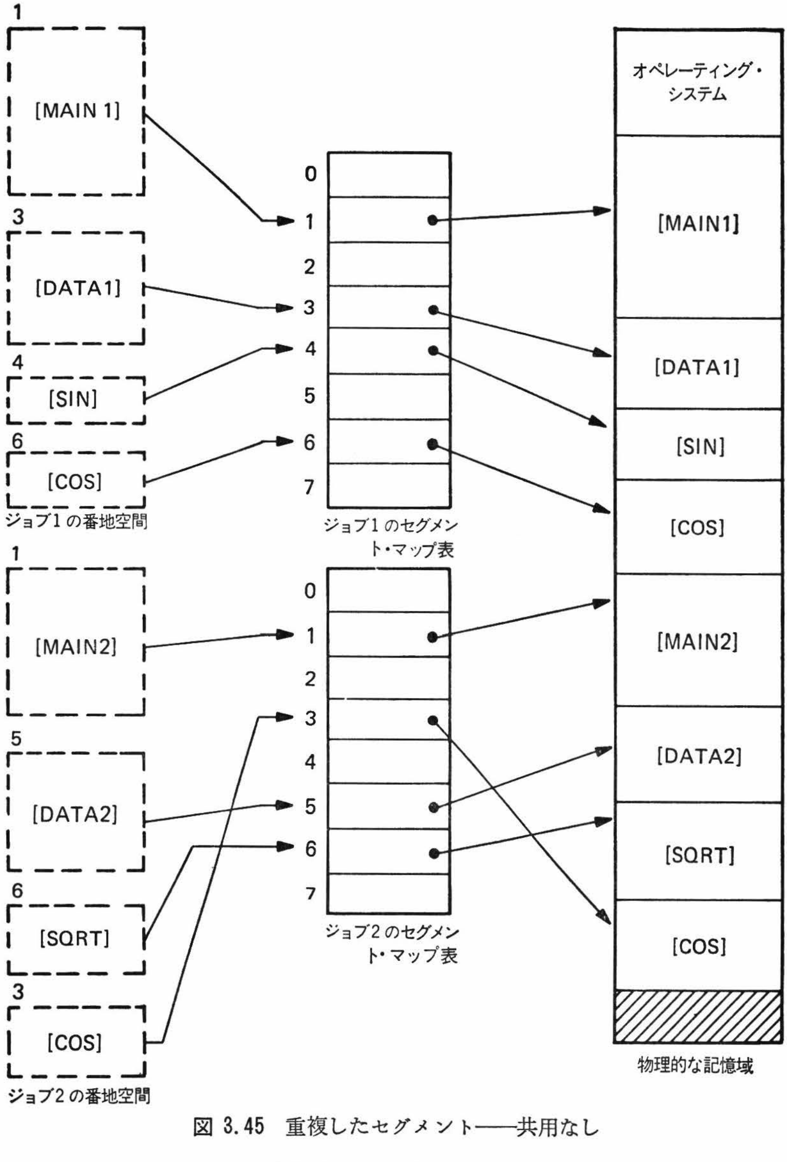


図 3.45 重複したセグメント——共用なし

トの中の変数だけを用いて動作する．純手続きは自分自身を書変えたり変更したりしない．したがって，純粋である手続きセグメントは安全に共用することができる．一方リンケージ・セグメントは各ジョブごとに独自に存在する．リ

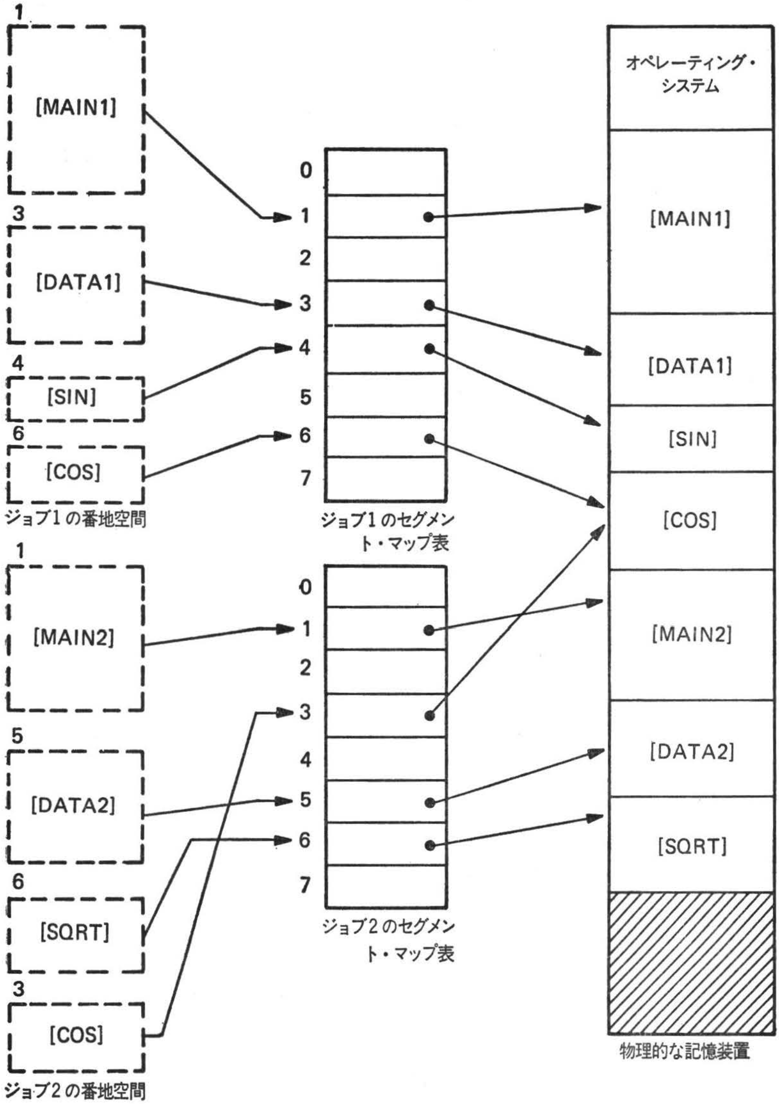


図 3.46 共用セグメントの例

ンケージ・セグメントは共用されることはなく，セグメント番号は，各ジョブごとに独自のリンケージ・セグメントの中にのみ表われるから，同一の手続きに異なったセグメント番号を割付けることが可能である．

セグメントの共用を調整するために，システムにおいて**アクティヴ・セグメント表**（AST）が使用される．1つのジョブが共用可能なセグメントを始めて参照すると，AST に1つの欄が設けられる．この欄は，セグメントの状態（すなわち，主記憶中にあるか否か），そのセグメントを使用しているジョブの番号，およびそのジョブに割付けられているセグメント番号を示している．したがって，ジョブが記号によって1つのセグメントを参照すると，重ねて複写しないようにするために，ダイナミック・リンカによって，セグメントがすでに AST に登録されていないかどうかの検査が行なわれる．そのセグメントがすでにアクティヴ・セグメント表に登録されていれば，セグメントの利用者一覧表にジョブの名前が追加され，同一セグメントを共用するようにセグメント・マップ表（SMT）が設定される．

AST の検査は，ジョブが最初にそのセグメントを参照するときだけ行なえばよい．それ以後の参照は，結合がすんだリンケージのための間接番地語を介して自動的に行なわれるか，あるいはアクティヴ参照表（ART）を使用して行なわれる．AST と ART の使用法とその目的の相違点を区別することは重要である．AST はシステム全体に1つだけ存在する．AST は，同一のセグメントを共用する目的で SMT の欄に情報を設定するために使用される．ART は各ジョブごとに存在する．この表は，ジョブにおいて1つのセグメントに同一のセグメント番号が使用されるようにしつつ，リンケージのための間接番地語を設定するのに使用される．

セグメントが主記憶から取除かれたときには，そのセグメントを共用しているジョブの SMT の欄にあるセグメント割出し表示子を設定しておかなければならない．AST は，これらの欄をすべて探し出すのに必要な情報を与える．

逆に，1つのセグメントが1つのジョブにより参照されて主記憶の中に再びロードされると，そのセグメントはこれを共用しているすべてのジョブからでも参照することができる．このように表の欄を調節する処理は，**間接番地指定**のハードウェアを使用することによって単純化することができる．SMT の各欄にある1ビットは，番地部が直接セグメントを指しているか（**セグメント直接指定語**モード），あるいはセグメントを指す番地語を指しているか（**セグメント間接指定語**モード）を表示する．後の場合には，AST の中にセグメントの番地を格納することができ，SMT の欄はすべて恒久的に適当な AST の欄を指すようにする．1つのセグメントが取除かれたときには，AST の欄を変更しておくだけでよい．

セグメント共用の組込みの詳細は複雑であるが，主記憶領域の利用では大幅

な節約となる．たとえば，10 のジョブで PL/I コンパイラを同時に使用していたとすると，共用しないときには別個の写しが必要である．普通の PL/I コンパイラは，手続きセグメントに 256K バイト，データと変数のためのデータ・セグメントに 32K バイト必要である．10個の写しでは10×(256K+32K)=2880Kバイト――約 300万 バイト――も必要である．手続きセグメントが共用されると，必要な記憶容量は 256K+10×32K=576K バイトに減り，80% 減少する．

3.7.2.6 制御されたアクセスの実施

各ジョブが別個の番地空間を持っている場合には，セグメント保護の機能によって，たとえば手続きセグメントを変更しようとしているといったプログラムの誤りを検出することができる．前述のような共用を大規模に行なうと，アクセスの制御の重要性が増す．純手続きを共用するときには，ジョブがセグメントを，事故にせよ故意にせよ，変更することを禁止しなければならない．さらに，通常はジョブが手続きセグメントを“読出す”ことを禁止する．このような制限を設ける理由は2つある：

1. 手続きセグメントを読出そうとすることは，普通，プログラムの誤りであることを意味する．
2. 手続きは**財産**（すなわち，誰かの“所有物”でプログラムの使用料を取っているもの）であり，“盗み出して複写”されるのを禁止しておきたい．

1つのセグメントが“実行のアクセス”だけを認めている場合，このセグメントは手続きとして呼出すことができるが，読出したり書込んだり（LOAD および STORE）することは禁止される．

手続きだけではなく，データ・セグメントも共用することができる．たとえば，主要都市と人口を納めた表を幾つかのジョブで同時に共用することができる．無意識でこの表を変更したりプログラムの誤りから保護するために，このセグメントは“読出しのアクセス”だけを認めるべきである．このセグメントに書込もうとしたり，プログラムとして実行しようとすることは禁止される．

アクセス制御の情報はセグメント・マップ表に格納されることから（図 3.39 参照），各ジョブに対しては同一のアクセスの制限である必要はない．株式市況を格納したデータ・セグメントがあったとしたら，1つのジョブが定期的に株価を更新するように**書込みアクセス**を認め，他のすべてのジョブは**読出しア**

セグメント＼利用者	Donovan	Madnick	Ziering	悪 者	・・・
SQRT	E	RWE	E	E	
PL/I	RWE	E	E	なし	
株価	RW	R	R	R	
給与	R	RW	R	なし	
・					
・					
・					

図 3.47 アクセス制御行列

クセスに限定することができる．各利用者に認めてよい適当なアクセスを決定する方法については第6章でより詳細に論じる．図 3.47 で示した概念的に簡単な方式では 一方の側にすべてのジョブ（または，利用者），他の側にすべてのセグメントを並べた行列を用いている．交叉する位置には，1人の利用者の特定のセグメントに対するアクセス権が表示されている．アクセス制御は多くの研究の主題となっている．広範囲の文献の要約が Graham (1972) および Hoffman (1969) によって行なわれている．

3.7.3 利 点

セグメンテーションの特徴は本節の始めで詳細に論じている．要約すると，

1. 断片化を解消する．
2. 仮想記憶を作る．
3. 動的にセグメントが拡張できる．
4. ダイナミック・リンキングが容易になる．
5. セグメント共有の機能を与える．
6. 制御されたアクセスを実現する．

これらの特徴のほとんどすべては記憶装置の利用率の改善に直接的に関与する．

3.7.4 欠　　点

セグメンテーションは，他の進歩的な記憶管理方式と同様の欠点を有している．ハードウェアの経費の増大，番地写像のための処理装置のオーバヘッド，表のための記憶域を追加する必要があること，およびオペレーティング・システムの複雑さが増大することなどである．このほかに次に述べる問題点が生じる：

1. 動的にセグメントが拡張することを認め，断片化を解消するためには詰め合せのオーバヘッドは大幅に増す．
2. 2次記憶上で可変長のセグメントを管理することは困難がある．
3. セグメントの最大長は主記憶容量で制限される．
4. セグメントのスラッシング（ページ・スラッシングと同様の現象）を防ぐには新しい技法を開発するか制限を設けなければならない．

3.8 セグメンテーションとデマンド・ページングによる記憶管理

セグメンテーションの持つ論理的な利点を活用し，欠点の多くを取除くための1つの方法は，セグメンテーションとページングの機構を組合せることである．各セグメントを1つの連続した実体として取扱う代わりに，各セグメントをページに分割することが可能である．1つのセグメント全体を取扱うのでなく，これらのページを物理的に処理すると詰め合せや2次記憶域の取扱い，およびセグメント長の制限の問題を取除くことができる．この手法はIBM 370および Honeywell 6180 によって採用されている．

3.8.1 ハードウェア

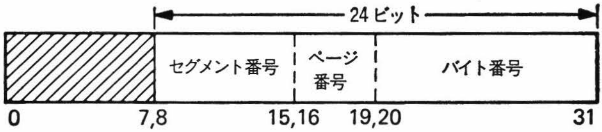
ハードウェアの組込みには多数の変形が存在する．本節では IBM 370 のハードウェアについて簡単に述べる．Honeywell 6180 は全く同様であり，Bensoussan (1969)，Daley (1968)，および Organick (1972) によって述べられている．

IBM 370の動的番地変換機能は，2K あるいは 4K バイトのページで，最大長 64K あるいは 1024K バイトのセグメントを可能にする．ページおよびセグメント容量の選択は，オペレーティング・システムで制御レジスタの設定を行なうことによってなされる．4K バイトのページで 64Kバイトのセグメントと

して設定されているときは次のようになっている.

24ビットの実効番地は3つの部分に分割される：

1. セグメント番号
2. ページ番号
3. バイト番号



したがって，ハードウェアは，最大16ページ（64Kバイト）のセグメントを最大256個まで取扱うことができる.

仮想番地から物理番地への写像は，図 3.48 に示したように，一連の表を用いて達成される．セグメント表レジスタ（制御レジスタ1）はSMTの番地と長さを示している．SMT の各欄は，セグメント割出し表示子が設定されていないときには，PMT の番地と長さを示している．PMT の各欄は，ページ割出し表示子が設定されていないときには，そのセグメントの各ページのブロック番地を示している.

ハードウェアの経費の節減と，各種の表のビット数を小さくするために，図 3.48 に示すように，ある種の制限が課せられる．たとえば，SMT の各欄の中の PMT の番地の部分には24ビットの番地が必要であるが，PMT を8の倍数の番地（すなわち，下位3ビットはつねに0である）から開始すると，SMT の欄には21ビットだけでよい.

表の各欄の空きビットは，SMT の欄の中の保護ビット，PMT の欄の中の参照／変更ビット，およびセグメント表レジスタの長さの部分を拡張してセグメント番号を増加させようとするときなど，将来ハードウェアの変更を行なうときに都合のよい手段となる.

SMT および PMT をアクセスするために生じる能率の大幅な悪化を防ぐために，アソシアティヴ・スクラッチパッド・メモリ（連想記憶装置）——**トランスレーション・ルックアサイド・バッファ**（TLB）——がハードウェアで自動的に使用され，SMT と PMT の参照のほとんどの部分を不要としている．一般に，連想記憶を使用すると番地変換のオーバヘッドを10%以下にすることができる.（これに関しては第 3.10.3 節を参照.）

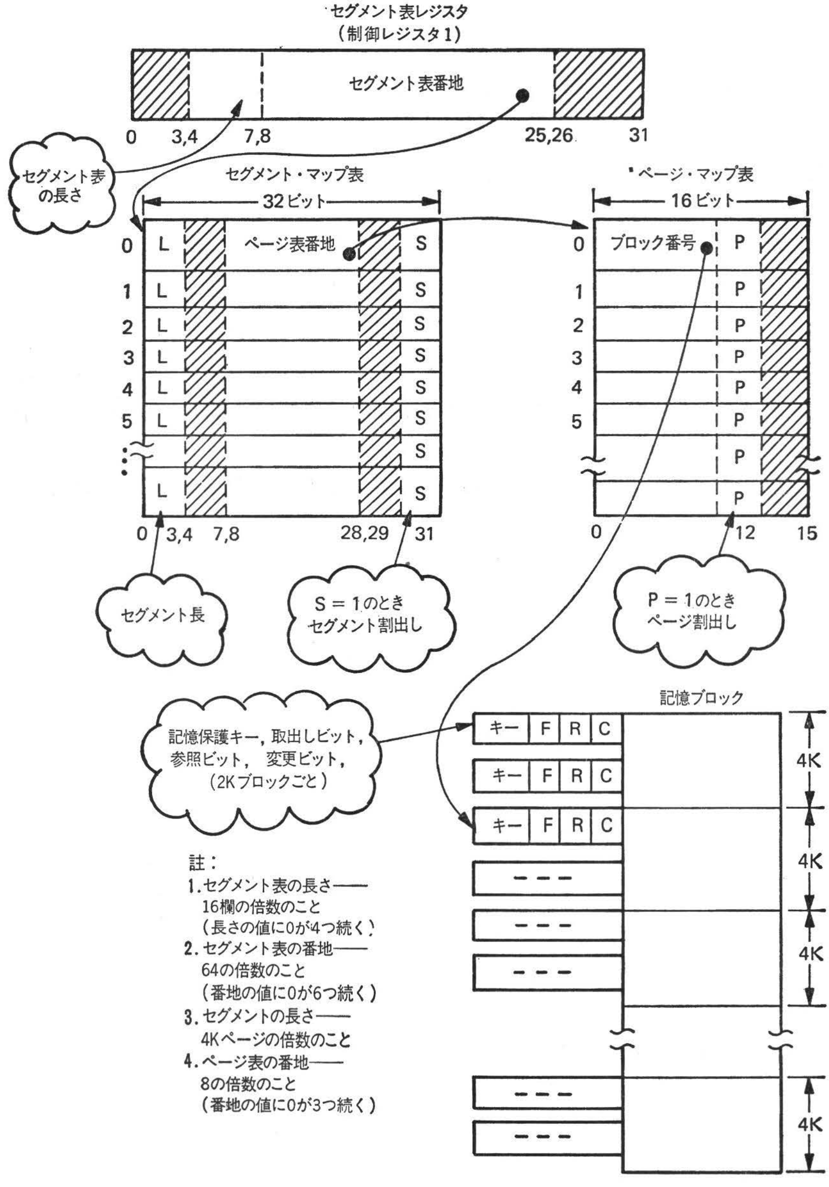


図 3.48 IBM 370 のセグメンテーション-ページングのハードウェア

3.8.2 ソフトウェア

デマンド・ページングとセグメンテーションの2つの記憶管理の節において示されたアルゴリズムは直接応用することができる．わずかに変更されるのは，両方に使用されるものを省くことである．図 3.49 は番地変換ハードウェアとソフトウェアのアルゴリズムとの間の相互の関係を示している．

たいていの番地参照においては，ソフトウェアの介入を必要とせずに変換が自動的に遂行される．各割込みの処理は次のように要約される：

1. **リンケージ割出し**　記号によるセグメント名にセグメント番号を割付け，SMT の欄の中のセグメント割出しビットを設定し，ART の中に1つの欄を作り，リンケージのための間接番地語にセグメント番号を設定する．
2. **セグメント割出し**　もしまだ欄が存在していないならば AST の中

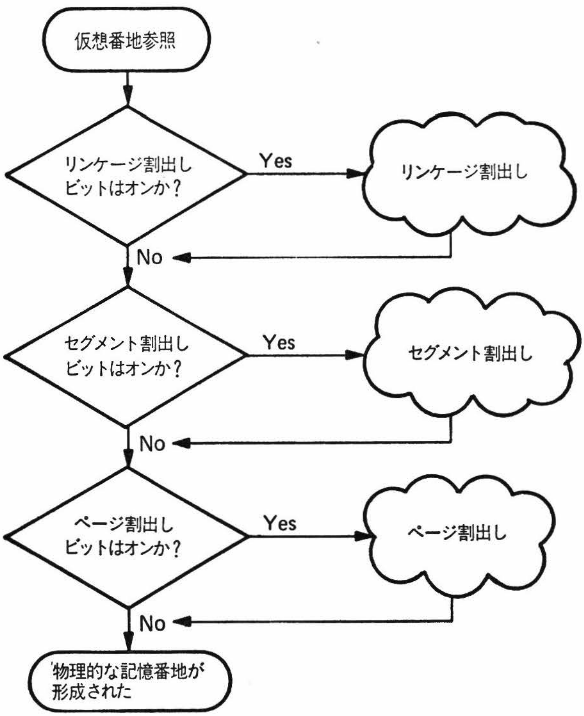


図 3.49 ハードウェアとソフトウェアの相互関係

に1つの欄を作り，ページ割出しビットをオンとしたPMTとファイル・マップ表をASTの欄の中に設定し，SMTの欄をPMTを指すように設定する.

3. **ページ割出し**　必要ならば1つのページを主記憶から取除いて空きブロックを求め，要求されたページをロードし，PMTの欄を適当に設定する.

セグメンテーション・デマンド・ページングの記憶管理の動作は多くの点で，第6章に述べる情報管理の動作ときわめてよく似ている.

3.8.3 利　　点

セグメンテーションとデマンド・ページングの組合せは，第3.6.4および第3.7.3節で示した利点のすべてを持っている.

3.8.4 欠　　点

セグメンテーション・デマンド・ページング方式において，ハードウェアの経費の増大，処理装置のオーバヘッド，およびシステムの複雑化はきわめて重大である.

さらに，ページ余りとSMTおよびPMTの記憶域の問題，デマンド・ページングとセグメンテーションの両方の場合に起こるスラッシングの危険性は依然残っている.

IBMのOS/VS2やHoneywellのMULTICSが成功していることから，これらの欠点は最小限にできることがわかる．セグメンテーション・デマンド・ページングによる記憶管理は，現在のところ，大規模で汎用の記憶管理に対する最も一般的で融通性のある手法である.

3.9 その他の記憶管理方式

これまでに述べてきた基本的記憶管理方式には多数の変形がある．本節では幾つかの主要な変形について論じる.

3.9.1 スワッピング

デマンド・ページングおよびセグメンテーションでは，それぞれページおよびセグメントが主記憶と2次記憶の間で交換された．この手法は他の記憶管理

技法に関しても採用することが可能である.

初期の MIT のコンパチブル・タイムシェアリング・システム (CTSS) では, 基本的な単一連続割付けの記憶管理方式を用いており, 一時にはただ1つのジョブだけが主記憶に入っていた. 短時間実行すると, 現在のジョブの番地空間全体は2次記憶にスワップされ, 別のジョブを主記憶に入れて実行できるようにする. (CTSS の記憶管理技法については第9章でもっと詳細に論じる.) 今日では小規模なタイムシェアリング・システムの多くはこの**スワッピング技法**を使用している.

同様にして, 分割割付けや再配置型分割割付けによる記憶管理システムの区画もスワップすることができる. IBM の OS 360 では, 高い優先順位のジョブの処理が必要となり, 十分な主記憶領域が得られず, 主記憶から取除いてもよいジョブが1つ以上存在するときには, そのジョブが主記憶から取除かれ2次記憶にスワップされる (この処理は**ロールアウト**と呼ばれる). 後刻, 高順位のジョブが完了すると, 低順位のジョブは再び主記憶にロードされる (**ロールイン**). IBM OS 360 のタイムシェアリング・システム (TSO) では利用者の区画をスワッピングして, CTSS のような機能を実現していた. 分割割付けを行なっている場合, 区画は取除かれたのと全く同じ位置にロードする必要があったので, ロールアウト/ロールインはぶざまであった. DEC PDP-10 や Honeywell 6000 シリーズのように再配置型分割割付けを行なっている場合には, 再配置ベース・レジスタが正しく設定されている限り, 区画は十分な容量の任意の空き領域にロードすることができる.

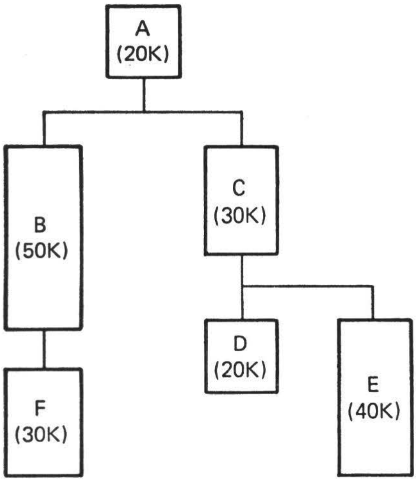
本節で述べたスワッピングは, ジョブの番地空間全体をスワッピングしなければならないので, あまりにも原始的である. もしジョブの容量が, タイムシェアリングの応用プログラム (エディタなど) でよく見られるように非常に小さいときには, スワッピングのオーバヘッドは小さくてすむ. 大きなジョブにスワッピングが行なわれるとオーバヘッドが大きい.

3.9.2 オーバレイ

上のスワッピング技法をもっと巧妙にして, ジョブの番地空間の一部分だけをスワップするのが**オーバレイ管理**と呼ばれるものである. オーバレイは, 通常, 単一連続割付け, 分割割付け, あるいは再配置型分割割付けの記憶管理において使用され, 基本的にはセグメンテーションに似たような機能を与える. しかし, この場合にはセグメントの番地写像のハードウェアはない.

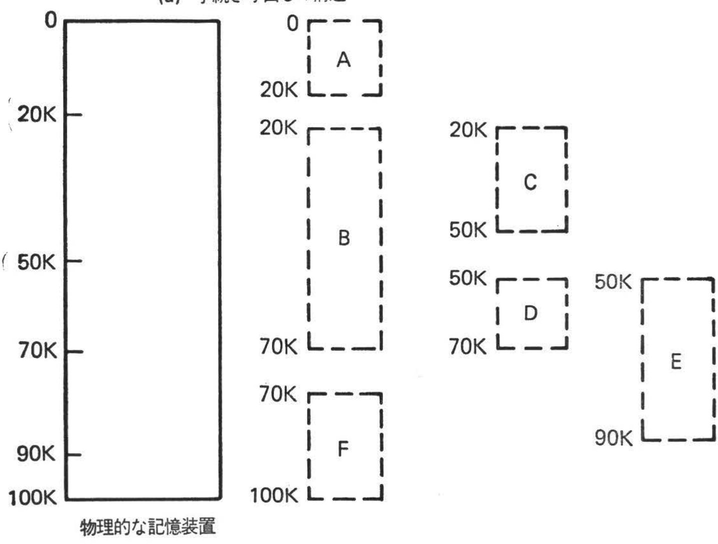
図 3.50 **a** はジョブの番地空間における手続きの関係を示している. この図

では，手続きAは手続きBとCを，手続きBは手続きFだけを，手続きCはDとEだけを，それぞれ，呼出している．この結果，BがCを呼出すことはない



番地空間の合計＝190Kバイト

(a) 手続き呼出しの構造



(b) オーバーレイの割付け

図 3.50 オーバレイの記憶管理

し，CがBを呼出すこともない．したがって，手続きBとCを同時に主記憶に入れる必要はない．図 3.50 b に示すように手続きの番地を割付けると，主記憶中でジョブの番地空間を収容するのに 190K バイトを使用する代わりに，最大 100K バイトだけが必要である．手続き間の参照が行なわれるときには，制御は特別の**オーバレイ・スーパバイザ**に移され，心要ならば，呼ばれた手続きをロードする．

IBM 1130 のディスク・モニタ・システムや OS 360 で使用されているように，たいていの自動的なオーバレイ・システムでは，利用者は前もってオーバレイ構造を宣言しておかなければならない．オーバレイのスワッピングは，互に排他的な手続に対する呼出しが交互に頻繁に行なわれるとき，はげしくなる．たとえば次のループにおいては，手続きAはBとCをそれぞれ 1,000 回スワップさせる：

```
DO I=1 TO 1,000;
CALL B(I);
CALL C(I);
```

3.10　将来の記憶管理

計算機システムの技術は急速に変化し，発展している分野である．本章で述べた記憶管理技法も間違いなく将来変化する．本節では3つの主要な話題について述べる：

1. より大容量の記憶装置
2. 階層化された記憶域
3. オペレーティング・システムの機能のハードウェア化

3.10.1　大容量の主記憶

多年にわたって主記憶の技術は多くの研究の主題であり，その結果経費は減少してきている．1960年代の中期から1970年代の初期の間に記憶装置の価格は元の 1/8 にも下がっている（たとえば，IBM 360/65 の記憶装置は1バイト当たり ＄1.60 であったが，新しい IBM 370/168 では1バイト当たり ＄0.20 である）．この10年間に価格は 1/10 に下がると予想されており，現在の実験的な技術を開拓することによって 1/100 に下がるという予想すら行なわれている．

このように価格が目ざましく低下すると，将来はきわめて大容量の主記憶が

経済的に利用可能となる．1,000万とか1億バイトの主記憶が普通のこととなるであろう．これらの発展は，より多くの主記憶領域が使用できるので，記憶管理の重要性が低下すると解釈されるかもしれない．しかし，筆者は，利用者にとって本当に十分大容量で低価格になるかどうか疑わしいと考えている．過去において，記憶容量が大きくなり，処理装置が高速になるにつれて，プログラマは前には不可能だったもっと大きくてもっと複雑なプログラムを書いたにとどまった．“知能的”な処理システムや人工知能における新しい応用の多くはもっと大きい記憶領域を必要とする．たとえば，英語で利用できる計算機や，テレビのようなカメラで物を“見て”その情報を処理する計算機ではどれだけの記憶域が必要となるであろうか．

しかしながら，過去の制限の多くは減少し，必要なハードウェアの構造がより明確にされることによって，もっと効果的で効率のよい記憶管理技法が生まれるであろうと信じている．

3.10.2 階層化された記憶域

前述した各種のページ・スワッピングやセグメント・スワッピングの技法は階層化された記憶域の利用例である．**記憶の階層**とは，速度と経費によって記憶装置を順序づけたもので，最も重要な情報だけが高速で高価な装置に収容され，最も重要でない情報が低速で安価な装置に収容される（Madnick, 1973）．“重要性”が変化するにつれて，情報は記憶の階層の中で移動する．人間が情報処理を行なう場合が記憶階層処理のよい例である．一般に，最も重要な情報を“頭の中に”入れている．やや重要度の落ちる情報はノートに記入されたり，ファイル・キャビネットなどに収容されたりする．図書館やその他の情報の保存場所は，現在のところ重要でない情報の全部を収容するための低速であるが安価な記憶装置として作用する．ある情報が突然重要になると，家で読むためにその本を図書館から借出す．その情報がかなり重要な場合には，その本を買うこともできるし，その情報を記憶することも可能である．

主記憶と高速の直接アクセスの2次記憶装置による自動的な2層の階層がページ・スワッピングやセグメント・スワッピング・システムで多く使用されている．図 3.51 は現在の記憶装置の技術で使用可能なものの価格と性能の範囲を示している．（これらの値は急速に変化しているので，表の値は比較のためだけで見てほしい．）最も速い装置と最も遅い装置の間では，性能に 10^7 の差があり，価格に 10^5 の差があることに注意してほしい．理想的には，これらの技術のすべてを含むような自動的な記憶階層を用いると，平均的には最も安価

装　置	乱アクセス時間（秒）	最大転送容量（バイト/秒）	価　格（$/バイト）	典型的な容量（バイト）
1. キャッシュ記憶 (IBM 3165)	1.6×10^{-7} (160 ナノ秒)	1×10^{8} (100Mバイト/秒)	8.8×10^{0} ($8.80)	1.6×10^{4} (16K)
2. 主記憶 (IBM 3360)	1.44×10^{-6} (1.44 マイクロ秒)	1.6×10^{7} (16Mバイト/秒)	5×10^{-1} (50¢)	5.12×10^{5} (512K)
3. 大容量記憶 (AMS SSU)	1.3×10^{-4} (130 マイクロ秒)	8×10^{6} (8Mバイト/秒)	8.8×10^{-2} (8.8¢)	2×10^{6} (2M)
4. 大記憶 (IBM 2305-2)	5×10^{-3} (5 ミリ秒)	1.5×10^{6} (1.5Mバイト/秒)	2.2×10^{-2} (2.2¢)	1.1×10^{7} (11M)
5. 大記憶 (IBM 3330)	3.8×10^{-2} (38 ミリ秒)	8×10^{5} (800Kバイト/秒)	4.5×10^{-4} (.045¢)	2×10^{8} (200M)
6. 記録用の大記憶 (Grumman MASSTAPE)	6×10^{0} (6 秒)	6×10^{5} (600Kバイト/秒)	2.2×10^{-5} (.0022¢)	1.6×10^{10} (16B)

図 3.51　記憶装置の種類（Madnick, 1973）

な（最も低速）装置程度の経費で，最も高速な（最も高価）装置程度の速度の大容量の仮想記憶を利用者に提供することができる．第 3.10.3 節は総合的な効率予測の手法を示している．

自動的な多層の記憶階層の組込みに関連する多くの問題を解決するために，多くの実験的および理論的な研究が依然必要である．これらの解決には，オペレーティング・システムの設計者，ハードウェア技術の専門家，および理論家の協力が必要である．この分野では将来多くの活動が行なわれるであろう．

3.10.3　記憶管理のハードウェア

計算機用の電子回路の価格が低下するにつれて，オペレーティング・システムのより多くの機能は経済的にハードウェアやファームウェアの制御の下におくことができるようになった．実験用の VENUS 計算機システムでは，セグメンテーションとデマンド・ページングのアルゴリズムの大部分はハードウェア／ファームウェアの制御の下におかれている．

IBM 360/85 や 370/155，158，165，および 168 などで用いられている**バッファ・メモリ**とか**キャッシュ・メモリ**とか呼ばれるものは，第 3.6 節で述べた

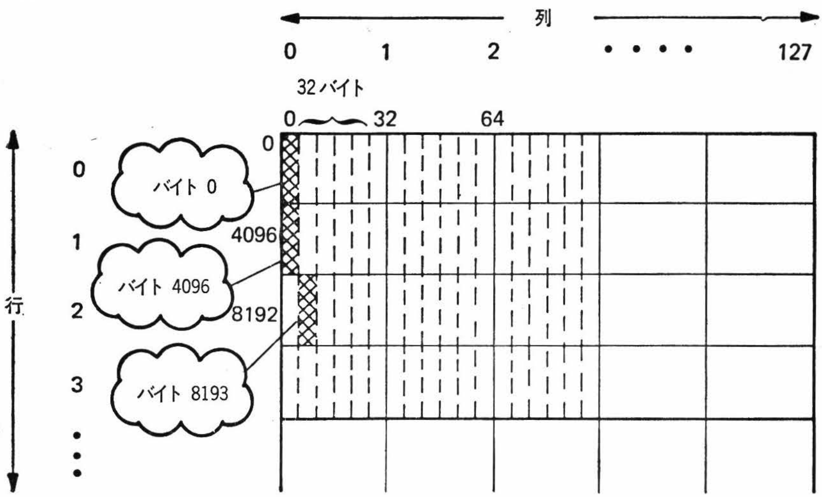
デマンド・ページングのハードウェア化である．主記憶の技術における速度と価格には調整が必要である．高速の主記憶（200ナノ秒以下で動作する記憶装置）は，図3.51に示したように，低速の主記憶（500ナノ秒以上で動作する記憶装置）に比べてはるかに高価である．

したがって，高速の，しかし高価な主記憶とするか，低速の，しかしずっと安価な主記憶とするかの選択をせまられる．**キャッシュ・メモリ**とか**バッファ・メモリ**とか呼ばれる小容量の高速記憶装置を，**主記憶**と呼ばれる大容量低速の記憶装置と組合せると，ほぼ高速記憶装置の速度で，安価な記憶装置の経費のハードウェアで仮想記憶システムを構成することができる．IBM 370で用いられている技法は以下の節で述べる（IBM, 9；IBM, 10；IBM, 11；IBM, 12；Liptay, 1968；Mattson, 1970)．

3.10.3.1 バッファ・ブロックの番地形成

記憶管理の議論において，番地写像の最終結果は**物理的な記憶番地**であると述べた．この番地は，指定された情報を取出したり格納したりするために記憶装置のハードウェアで使用される．ハードウェアはこの番地をどうするのであろうか．記憶装置の典型的な動作は第5.2.2節において論じられる．本節で論じるシステムでは，すべての情報は32バイトのブロック（記憶ブロックという）に物理的に詰め込まれている．ハードウェアが1つのバイトを変更しなければならないときは，このバイトを収容している32バイトのブロックが取出され，バイトが変更され，そしてそのブロックは記憶装置に返却される（しばしば，**読み書きサイクル**と呼ばれる)．さらに，これらのブロックは，それぞれ 4096バイト（128 ブロック）の**行**に編成される．1つの行の中のブロックには**列番号**がつけられている．図 3.52**a**はこの行，列，バイトの編成を示している．

したがって，物理的な記憶番地は，図 3.52**b**に示したように，実際には記憶装置のハードウェアによって，行番号，列番号，およびバイト番号に分割されて物理的な情報の参照が行なわれる．この変換は，すべて記憶装置のハードウェアによって自動的に行なわれ，オペレーティング・システムや利用者プログラムにとってはわからない．(“ハードウェアをよく知っている人”への註釈：ここでいう行や列は必ずしも，記憶装置のハードウェアのもっと基本的で物理的な**記憶バンク**や**インターリーブした記憶装置**には対応しない．これらの話題については本節では論じない．）行の個数は主記憶容量によって決定される．



(a) 行，列，およびバイトによる物理的な記憶番地の指定



（たとえば，バイト8193は行 2，列 0，バイト1である）

(b) 行，列，およびバイト番号で示された物理的な記憶番地の解釈

図 3.52 行，列およびバイトの番地指定

3.10.3.2 バッファ記憶装置

バッファ記憶システムは**キャッシュ記憶システム**とも呼ばれ，特殊な**バッファ記憶制御装置**が処理装置と主記憶との間に置かれる．バッファ制御装置は，図 3.53 a に示したように，小容量の**バッファ・メモリ**と**バッファ内容の指標配列**の 2 つの主要部分から構成されている．

簡単なバッファ・メモリは，図 3.53 b に示したように，主記憶と同様 32 バイトのブロックに分割されている．主記憶には複数の列が存在するので，バッファ・メモリには多数のブロックが存在する．この例では 128 のブロックが存

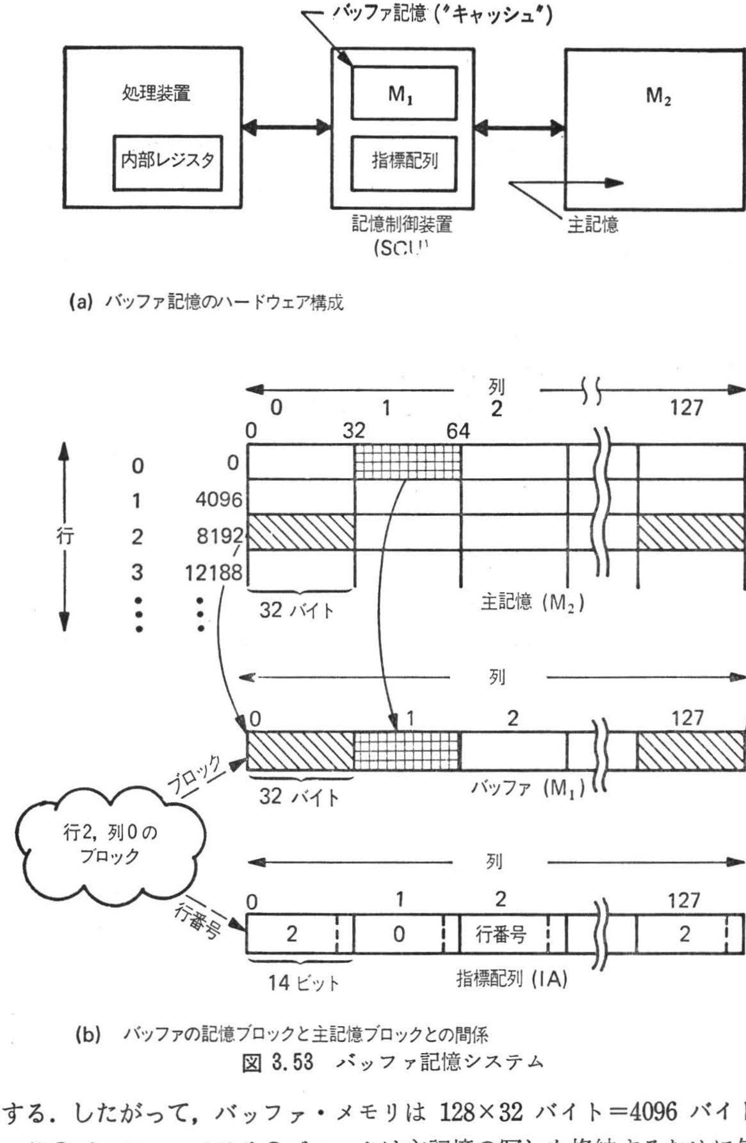


図 3.53　バッファ記憶システム

在する．したがって，バッファ・メモリは 128×32 バイト＝4096 バイトである．このバッファ・メモリのブロックは主記憶の写しを格納するために使用される．指標配列は，バッファ・ブロックの写しがどの主記憶の行からきたものかを表示している．図 3.53b では，主記憶 M_2 の 3 つのブロックは次のとおりである：

1. 32〜63番地（行 0，列 1）
2. 8192〜8229番地（行 2，列 0）
3. 12156〜12137番地（行 2，列 127）

ブロックの複写が M_2 から行なわれるときには，このブロックは M_1 の対応する列に格納される．指標配列は対応する行番地を表示する．

3.10.3.3 バッファ管理のアルゴリズム

バッファ・メモリの管理のための1つのアルゴリズムが図 3.54 に示されている．読出し要求が出されたときには，要求されたバイトがバッファ・メモリ M_1 のブロックに入っていないかどうかの検査が行なわれる．入っていれば，主記憶 M_2 をアクセスする必要はない．M_1 に入っていないときには，このブロックは M_2 から取出され，M_1 の適当なブロックに格納される．このアルゴリズムは単純であり，すべてをハードウェアで組込むことが経済的に可能である．

書込み要求によってバッファ・ブロックが変更されたときには，M_2 の対応するブロックが更新される．これを**ストア・スルー**という．主記憶中の写しはつねに同一内容であるので，任意のブロックは任意の時点で置換えることができる．

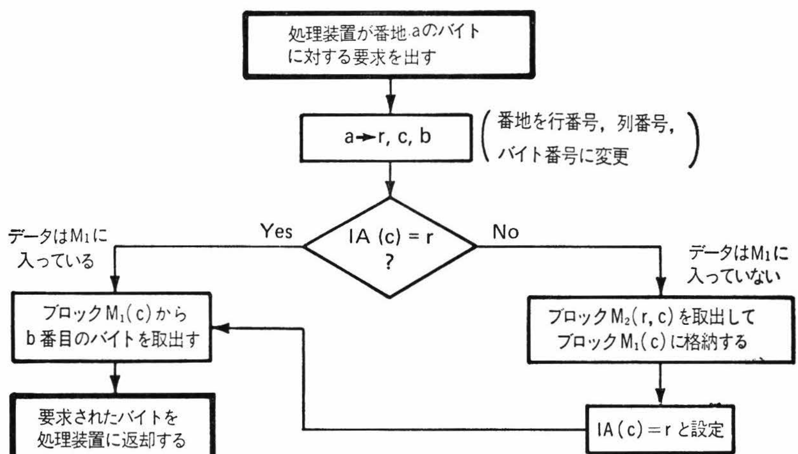


図 3.54 バッファ管理のアルゴリズム

バッファ・メモリのアルゴリズムの動作を説明するために次の短いプログラム・セグメントを考えてみよう：

	番地	記憶内容	
		⋮	
	64	00000017	データ語 4バイト長
	68	⋮	
ここから開始 →	4096	ADD 1, 64	命令
	4100	GOTO 4096	4バイト長
	4104	⋮	

このプログラムは，64番地の内容（数17）を処理装置のレジスタ1に連続的に加算する小さなループを実行する．このプログラムには何も変わった点はないが，多数の重要な点を示している．実際，このプログラムは，次のような配列処理プログラムと，インデック指定およびループ終了条件がないだけで，機能的には全く同一である：

```
SUM=0;
DO I=1 TO 100;
SUM=SUM+A(I);
END;
```

上のプログラムは，次のような処理装置の参照要求を生じる：

1. 4096番地から命令を取出す（ADD 1,64）.
2. 64番地からデータを取出す.
3. 4100番地から命令を取出す（GOTO 4096）.
4. 4096番地から命令を取出す.
5. 64番地からデータを取出す……

最初 M_1 が空であり，図 3.54 のバッファ管理アルゴリズムの使用を仮定すると次に示すことが起こる：

1. 4096番地から命令を取出す.

a．4096→$r=1$，$c=0$，$b=0$.

b．IA(0)≠1（最初 M_1 は空である）.

c．ブロック $M_2(1, 0)$——4096～4127番地——が M_2 から取出され，$M_1(0)$ に格納される.

d．IA(0)=1 に設定する.

e．取出し要求は完了する.

2. 64番地よりデータを取出す.

a．64→$r=0$，$c=2$，$b=0$.

b．IA(2)≠0（初期条件）.

c．ブロック $M_2(0, 2)$——64～95番地——が M_2 から取出され，$M_1(2)$ に格納される.

d．IA(2)=0 に設定する.

e．取出し要求は完了する.

3. 4100番地から命令を取出す.

a．4100→$r=1$，$c=0$，$b=4$.

b．IA(0)=1（1d による）.

c．IA(0) のバイト4～7によって命令を実行する.

d．取出し要求は完了する.

4. 4096番地から命令を取出す.

a．4096→$r=1$，$c=0$，$b=0$.

b．IA(0)=1（1d による）.

c．IA(0) のバイト 0～3 によって命令を実行する.

d．取出し要求は完了する.

5. 64番地からデータを取出す.

a．64→$r=0$，$c=2$，$b=0$.

b．IA(2)=0（2d による）.

c．IA(2) のバイト 0～3 からデータを抽出する.

d．取出し要求は完了する.

1と2の参照は M_2 をアクセスする必要があったが，3，4，5，…の参照は M_2 だけ参照すればよいのに注目してほしい．たいていのプログラムでも似たようなものである.

3.10.3.4 効率の解析

バッファ・メモリを使用するとどれ位の効率改善ができるであろうか．第

3.6 節（デマンド・ページングによる記憶管理）で述べたように，参照の局所性のために，要求されたデータは M_1 に格納されていることが多く，M_2 を参照する必要がなくなる（すなわち，たいていの参照は，図 3.54 の“データは M_1 に入っている”ときの径路をとる）．経験およびシミュレーションによる解析によって，90%以上もの参照はバッファ・メモリ M_1 を参照するだけで満足させることができる．この割合いは**成功率関数**と呼ばれ，第 3.6 節で定義された．この成功率関数の実際の値は実行されるプログラムとバッファ管理のアルゴリズムに関係する．

バッファ・メモリと主記憶とを結合した階層は，特性が静的に定まっている仮想記憶とみなすことができる．興味あるのは次の事柄である：

1. 実効容量　S
2. 1バイト当たりの実効経費　C
3. 1バイト当たりの実効アクセス時間　T

これらの特性は次のようにして計算することができる：

$$S' = S_2$$

$$C' = \frac{S_1 \times C_1 + S_2 \times C_2}{S_2}$$

$$T = f_1 \times T_1 + f_2 \times T_2$$

ただし，

S_1 は M_1 の容量
S_2 は M_2 の容量
C_1 は M_1 の1バイト当たりの経費
C_2 は M_2 の1バイト当たりの経費
T_1 は M_1 の平均アクセス時間
T_2 は M_2 の平均アクセス時間
f_1 は M_1 に対する成功率関数
$f_2 = 1 - f_1$

報告によると，IBM 370/158 では平均的に約 90% のバッファ成功率を得ている．これに対する実効的な効率は次表のように求められる：

	M_1	M_2	M'
容量(S)	8K	1M	1M
バイト当りの価格(C)	$4.00	50¢	53¢
アクセス(T)	230ナノ秒	1マイクロ秒	307ナノ秒

したがって，仮想記憶全体 M' としての実効的な効率は，M_2 の容量と M_2 程度の経費で，M_1 程度のアクセス時間となっていることがわかる．これらの効率の比較は 図 3.55 にはっきりと示されている．

3.10.3.5　その他の問題点

簡単のために，バッファ記憶システムの説明では多くの問題点が簡略化してある．詳細に関しては適当な参考文献を参照してほしい（Lipty, 1968；Mattson, 1970；Wilkes, 1965；Madnick, 1973）．

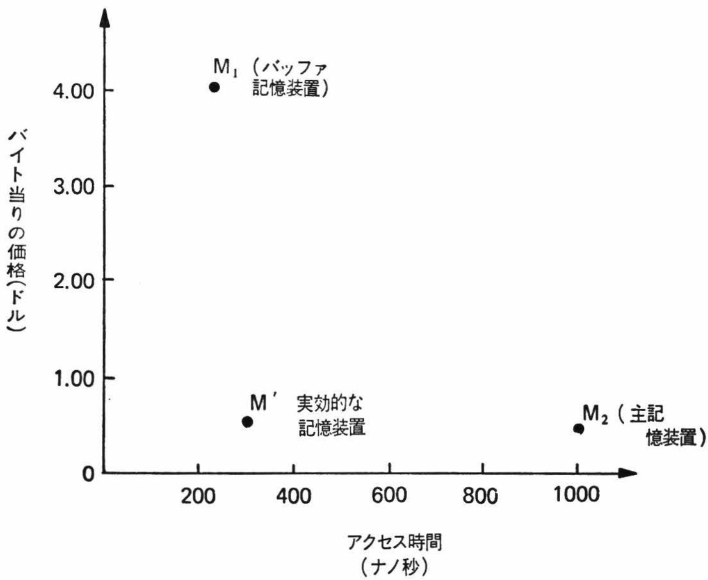


図 3.55　記憶装置の性能比較

バッファ記憶ブロックを主記憶ブロックに割付けるアルゴリズムでは，1つの記憶ブロックはつねに同一のバロッファ記憶ブックに割付けられた（すなわち，同一の列番号のものに割付けられた）．この方法は，**完全指定の連想的写像**と呼ばれる．この技法は単純で簡潔であり，時間のかかる探索が不要であるが，欠点もある．たとえば，行番号に関係なく，一時には1つの列だけが M_1 に入り得る．例題のプログラムのページ 000 が次のように変更されたとすると，命令（$r=1$, $c=0$）とデータ（$r=0$, $c=0$）とは同じ列になり，$M_1(0)$ に格納しなければならないようになる：

```
......
4096 ADD 1,0
4100 GOTO 4096
```

データ参照が行なわれるたびに $M_1(0)$ の中の命令の写しがデータの写しで置換えられ，命令参照のときはデータが置換えられる．したがってループを繰返すたびに2回 M_2 が参照され，M_1 は全く無用となる．この問題は**バッファの衝突**と呼ばれる．

このバッファの衝突に対する1つの解決法は，それぞれ独立の指標配列を備えた幾つかの M_1 を用いることである．このようにすると，同一の列から幾つかのブロックを同時に格納することができる．バッファ衝突の確率は 128（列数）の内のほぼ1つにすぎないので，この問題点を小さくするには，列当たり2～4個の M_1 ブロックを用意するだけでよい（極端な場合を除く）．この方法は図 3.56 に示されている．

IBM 370/158 では列当たり2個のバッファ・ブロックを持ち，もっと能力の大きい 370/168 では4個持っている．列当たりに複数のブロック（図 3.56 では a および b で示している）を使用することになると，**バッファ置換え方法**という新しい問題が生じる．たとえば，4096番地（$r=1$, $c=0$）が参照されたときにこのブロックが M_{1a} か M_{1b} に入っていないとしたら，どちらのブロック（a または b）を M_2 から取出す 4096 番地のブロックと置換えたらよいかを決定しなければならない．この問題を解決するには通常 LRU（最近最も使用されていないものを選ぶ）方式が使用される．

3.11 要　　約

本章では各種の記憶管理方式を述べてきた．各方式は，複雑さとオーバヘッ

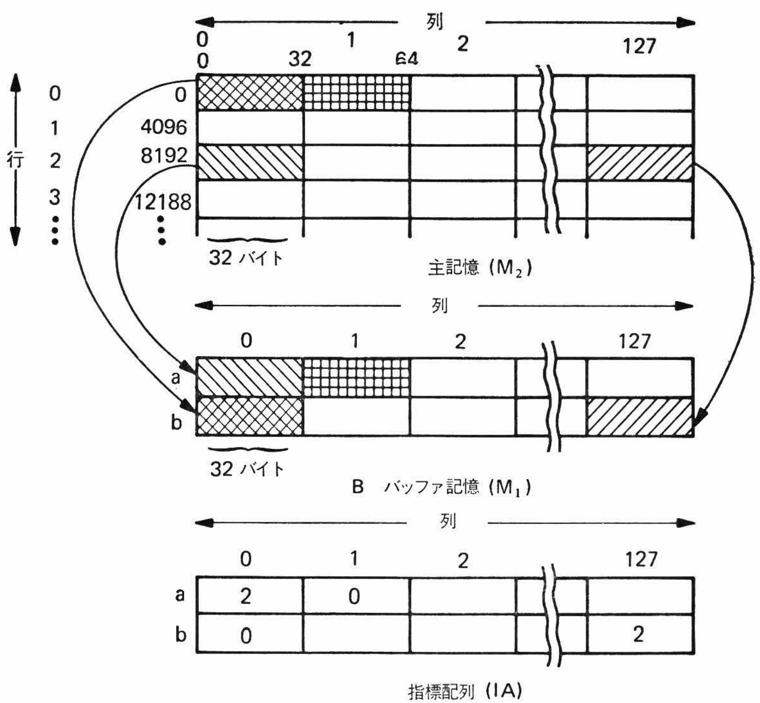


図 3.56 列ごとに複数のバッファ・ブロックを置く

ドは増すが，その前に述べた方式の利点をすべて有し，欠点の一部も解決した．

記憶管理は今日のオペレーティング・システムにおいて最も重要な問題であり，多くの研究の対象となっているので，この問題にかなりの時間をさいた．

演 習 問 題

3.1 次に示す分割割付けアルゴリズムの1つに対して要求記憶量と完了時間の関係が最適であり，他の2つではそうでないようなジョブの順番を作り出せ：

1. 静的な分割
2. 最適適合
3. 第1適合

3.2　分割割付けの記憶管理のアルゴリズムとして最適適合と第1適合の2つを述べた．別の“最悪適合”のアルゴリズムが提案されている．最悪適合は，つねにジョブに最大の区画を割付けようとするものである：

1.　その組込みについて述べよ．

2.　予想される効率について述べよ．

3.　最悪適合が最適適合や第1適合よりもよいようなジョブの順番を示せ．

4.　最悪適合の方が悪いようなジョブの順番を示せ．

3.3　次の仮定を行なって問に答えよ：

1.　ジョブの平均的な容量は256,000バイトである．

2.　1,024,000バイトの記憶容量がある．

3.　分割割付け方式が用いられると断片化のために1/3の記憶域が無駄となる．

4.　それに対して，再配置型分割割付け方式が用いられると区割容量の半分が無駄になる．

5.　入出力待合せ時間は70%である．

6.　平均的なジョブ実行時間は10秒である．

7.　詰め合せは毎秒1,000,000バイトの割合で実行できる．

8.　詰め合せのオーバヘッドは1ミリ秒である．

問題

a．詰め合せの料金を支払う必要があるか．その理由，たとえば記憶域の無駄，システムの入出力待合せなど，を述べよ．

b．aの答えが記憶容量で変化するとしたらその容量を示せ．

c．aの答えはジョブの実行時間で変化するとしたらその時間を示せ．

d．再配置型分割割付方式が用いられると，再配置のハードウェアによる番地計算のために実行が5%遅くなる．この場合a，b，cの答えはどのように変わるか．

3.4　分割割付けの記憶管理に対して別のアルゴリズムが提案されている．このアルゴリズムでは，すべての空き領域は番地の順に分類されている（第1適合と同様）．小容量の記憶域，たとえば32K以下，が要求されると，提案のアルゴリズムは空き領域の表を下から探す．大容量の記憶域が要求されると，このアルゴリズムは表の上から探す．

1.　この技法が第1適合や最適適合よりもはるかに優れているようなジョ

ブの順番を求めよ．

2. この技法の特徴を論じよ．どのようなとき優れており，どのようなとき劣っているのか．

3.5 本書では，再配置型分割割付け記憶管理技法はジョブに1つの再配置レジスタがあるとして述べた．

1. 2つの再配置レジスタがあればどんな利益があるか．
2. 2つの再配置レジスタを使用するのに処理装置の番地生成処理を変更する必要があるか．その理由と方法について述べよ．

3.6

1. IBM 360において，すべてのプログラムが（再配置型分割方式のように）再配置できるように，ソフトウェア上の規約と制限を設けよ．
2. この計算機で上のような規約を設けずに再配置することが可能か．可能ならば例を示せ．

3.7

1. 多重プログラミングとは何か．それは何故に使用されるのか．
2. 断片化とは何か．これはどのようにしたら克服できるか．
3. ページとセグメントの相違点は何か．
4. 純手続きとは何か．純手続きを使用するとどのような利益があるか．

3.8

1. 各種の記憶管理方式に対して，簡潔に主たる欠点をあげよ．
2. セグメンテーション・ページング割付けが“最良の”記憶管理技法であると考えられているが，この方式が適当でない場合も多い（すなわち，4Kのミニコンピュータでは MULTICS のような方式をとらないであろう）．4つの記憶管理技法について，その方式が他の方式よりも都合のよいような状況を述べよ．

3.9 大規模な多重プログラミング・システムについて論じるときに，記憶管理の1つの重要な機能はプログラムの共用の考えに関係していることを見出すであろう．特に，4つのプロセスが同時に PL/I のコンパイルを行なっているようなことを想定するとよい．PL/I コンパイラの物理的なコードは各プロセスとも同じであるが，“作業域”はそれぞれ異なっている．

PL/I コンパイラのような複雑なプログラムでは必要とする記憶容量も大きい．4つのプロセスがコンパイラの命令の入った物理的な記憶域を共

用できるようにすると都合がよい．このような共用が可能となるようにコンパイラを作成するにはどのようにすればよいかを次の各場合について述べよ：

1. IBM 360/370 のような分割割付けシステム
2. PDP-10 のように2つの再配置レジスタを持つ計算機
3. IBM 360/67 のようなページング・システム
4. MULTICS のようなセグメンテーション・システム

3.10 本章では7つの記憶管理技法を論じた．これらの方法は，記憶管理の問

記憶管理方式	解決した問題点	新しく必要となるハードウェア	新しく必要となるソフトウェア
単一連続割付け	——	——	——
分割割付け			
再配置型分割割付け			
ページング			
デマンド・ページング			
セグメンテーション			
ページングを行なうセグメンテーション			

題の処理にはその柔軟性に種々あり，問題を解決する手法の複雑さも様々であることを示している．これらは次の表に簡単なものから順に並べられている．それぞれの方法について，上の方法から次の方法になると解決できる問題と，その際必要となる新しいハードウェアおよびソフトウェアの機能を表に記入せよ．

3.11

1. 利用者の番地空間を定義せよ．
2. 次の場合の記憶域への写像を説明せよ．
   a．単一連続割付け
   b．再配置型分割割付け
   c．デマンド・ページングを行なうセグメンテーション

3.12

1. 複数の再配置および限界レジスタを持つ計算機が与えられたとする．多重プログラミングでページング方式を組込むにはどのようにすればよいか．
2. ただ1ビットの保護キーを使用して，任意の数の多数の利用者が使用する多重プログラミングを組込むにはどのようにすればよいか（利用者相互間の保護も行なわねばならない)．

3.13 デマンド・ページングにおいて，各ページに付属している“変更ビット”の果たす役割りを説明せよ．

3.14 単純ページングの場合を考えよう．

1. 記憶容量がMバイト，平均的な区画の容量がPバイト，およびページの容量がNバイト（ページ・マップ表の1つの欄の長さnは$\log_2 N$ビットである）とすると，単純ページングのための最適なページ容量はいくらか（すなわち，ページ余りとページ・マップ表との和)．答えはMおよびPにて表わせ．
2. $M=512$Kバイト，$P=128$Kバイトの場合の値を求めよ．
3. 問2の答えは，ページ・マップ表の欄の長さがバイトの倍数でなければならないとしたら（すなわち，$n=\dfrac{\log_2 N}{8}$ 小数部切上げ）変化するか．この制限が問題となるのはどのような場合か．

3.15 チャネルでは，すでに論じたように，チャネル・コマンド語(CCW)に

ェ実際の物理番地を記入する必要がある．入出力チャネルのためにハードウアの変更を行なわずにページングの記憶装置を使用するとしたら，1つの問題が生じてくる．それは，入出力動作を行なおうとしている最中にそのページが取除きの対象となるかもしれないことである．これには2つの解決方法がある：

1. 一時的にバッファを記憶域中で固定し，CCW を実番地に変更する．すなわち，ページングのソフトウェアでバッファのページを取除きの対象としないようにする．
2. バッファをオペレーティング・システムの(ページングされていない)一時記憶域の中に収容し，入出力が完了したとき読込みの場合にはデータを仮想記憶の中のページに移す．

問 題

a．1の方式を組込むためにはどのようにすればよいか．どのデータ・ベースやルーチンを変更しなければならないか．固定すべきページをどのようにして決定するか．

b．利用者のバッファがページの境界にまたがっており，2つのページが物理的に連続でないとしたら，どのようにして1の方式で読込み動作を遂行することができるか．

c．2の方式では，実際の記憶領域はページングを行なっている部分とオペレーティング・システムの領域に分割されている．利用者が入出力を要求したときオペレーティング・システムにはバッファのための領域がないとしたら，その領域が開放されるのを待たねばならないか，あるいは，オペレーティング・システムがページングの領域をバッファとして使用できるようにすることができるか．後者の場合にはどのようにしてこれを行なうか．この場合の問題点は何か．

d．どちらの方式が優れているかを決定する要素は何か．各方式について利点と欠点を1つずつあげよ．

e．チャネルで仮想記憶における CCW の処理が可能であれば，入出力処理は簡単になるか．簡単になるとすれば，その理由と方法について述べよ．

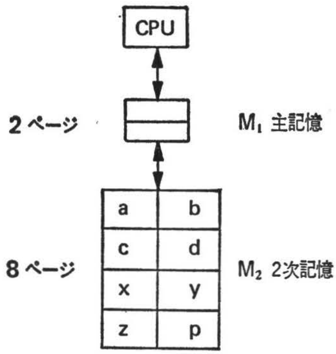
3.16 本問題では，デマンド・ページングにおいてページ容量を変更したときの効果について調べてみよう．

460 語のプログラムにおいて次のような順序で記憶域が参照されるものとする：

10, 11, 104, 170, 73, 309, 185, 245, 246, 434, 458, 364

1. ページの容量が100語であるとし，プログラムには200語の主記憶領域が使用可能で，FIFO の置換えアルゴリズムが用いられるものとする．この順序に対するページの軌跡を示せ．このページの軌跡に対する成功率を求めよ．
2. 次にページ容量を50語とし，ページ番号を 1′, 1″, 2′, 2″,… としてページの軌跡および成功率を求めよ．
3. ページ容量を200語として同様に行なえ．
4. 上の結果について考察せよ．求めた結果がページ容量が半分になったり2倍になったりしても成立すると思うか．
5. 小さいページを使用することは利点があるか．それを相殺する要素は何か．200語の情報を主記憶に転送するには，2次記憶装置の動作特性によって，100語を転送する時間の2倍より短い時間ですむことを留意せよ．
6. 主記憶容量が400語であるとして問1から3に答えよ．
7. このプログラムにより多くの領域が使用可能になったときはどのような効果があるか．このような効果がいつも予想されるか．その理由は何か．実際のプログラムのように，ページの軌跡がもっと長いとしたらどのような変化が予想されるか．
8. 本問題で調べたようなページの軌跡は実際のプログラム実行では起こりそうにもない．その理由は何か．同様な条件の下では，実際のプログラムのページ成功率はこの例に比べてもっと高いと考えるか，もっと低いと考えるか．

3.17 次のようなハードウェアがあるとする：



ページング方式は，必要なとき M_2 から M_1 にページを入れ，M_1 の領域が必要になると M_1 から M_2 にページを出す方式である．

次の表は，M_1 のページ数が2のとき FIFO の置換えアルゴリズムでページ軌跡を求めたものである．

ページ軌跡 P	a	a	b	b	c	c	a	b	
ページ取出しF	*		*		*		*	*	F＝5
M1の内容	a	a	b	b	c	c	a	b	
	–	–	a	a	b	b	c	a	

この表で最も重要な情報は，ページ割出しの発生回数である．成功率 S は，1からこの割出し回数を全参照回数（ページ軌跡の長さ）で除した値を減じて求められる．

$$f=\frac{F}{P}=\frac{5}{8}=0.625$$

$$S=1-f=0.375$$

以下ではページ軌跡を P と記す．

$$P=abacabdbacd$$

1. ページ軌跡 P に対し，FIFO の置換えアルゴリズムを用い，主記憶容量が2ページであるとして，ページ軌跡の表を作れ．このページングの成功率はどれだけか．

2. LRU アルゴリズムを用いて問1に答えよ.
3. FIFO の置換えアルゴリズムでは1回余計にページ割出しが起こるがその理由を説明せよ. 言葉を換えるとページ軌跡Pのどの特性によってFIFO より LRU の方が効率がよいか.
4. M_1 の容量が3ページであるとして問1に答えよ.
5. 問4の場合に何が起こったかを簡単に説明せよ. M_1 の容量はSにどのような影響を及ぼすか. この結果は一般的な傾向か. M_2 の容量を変化させるとSにどのような効果があるか.
6. FIFO および LRU の置換えアルゴリズムは直観的によい方法と考えられる. MRU (最も最近使用したものを取除く) 置換えアルゴリズムは直観的に悪いと考えられる. MRU 置換えアルゴリズムを用いて問1に答えよ. この結果から, 直観とPのようなページ軌跡の特性に関して何がわかるか.

3.18

1. 多層の記憶システムに関する成功率は実行中のジョブの混合体に関係する. その理由は何か. また, その依存度を最小にするにはどうすればよいか.
2. 多層の記憶システムで成功率を改善するためにはどのような技巧がハードウェアとソフトウェアで用いられるのか. プログラムのどのような性質が利用されるのか.
3. 個々の装置に対して, アクセス時間, バンド幅, 経費, 容量, および成功率が与えられているとする. これらを用いて次のようにして構成される多層の記憶システムのアクセス時間, バンド幅, 経費, および容量はどのようにして求めることができるか.

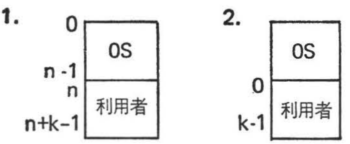
CPU$\cdots\cdots M_1\cdots\cdots M_2\cdots\cdots M_3$

$f_{12}=M_1$ に対する成功率
$f_{23}=M_2$ に対する成功率
$B_1, B_2, B_3=$それぞれ, M_1, M_2, M_3 に対するバンド幅
$t_1, t_2, t_3=$それぞれ, M_1, M_2, M_3 に対するアクセス時間
$C_1, C_2, C_3=$それぞれ, M_1, M_2, M_3 に対する1ビット当たりの経費
$S_1, S_2, S_2=$それぞれ, M_1, M_2, M_3 に対するビット容量
$B=$全体のバンド幅
$t=$全体のアクセス時間

C＝全体の１ビット当たりの経費

S＝全体の容量

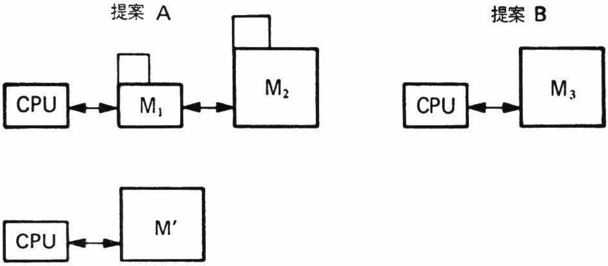
3.19 仮想記憶の議論においては，利用者の番地空間という観点から考察した．次に利用者の番地空間に対する２つの観点を示す．第１は，利用者の番地空間はオペレーティング・システムを含むとするもの，第２は，利用者プログラムの領域だけを含むとするものである．１つの番地空間の中に領域をとっても必ずしもその利用者にアクセス権を与える必要はない．IBM の OS/360 では，利用者の番地空間は，システム中の他の利用者と同様に，オペレーティング・システムを含んでいる．しかし，このようにしても保護キーによって，利用者は他の領域を侵害することが不可能である．



1. 第２の方式では，利用者はどのようにしてオペレーティング・システムと通信（すなわち，呼出し）を行なうか．

2. オペレーティング・システムに対して，番地を用いて引数を渡すとき（たとえば，利用者が指定したバッファ領域にカードを読込む），２つの方式の相違点を述べよ．

3.20 IBM 社（Intersteller Blundering Manufacturer）では計算機を開発して製造することに決定した．２つのグループがシステムを開発し，記憶システム以外はほとんど同一のシステムを構成した．グループＡは自動的な２層記憶システム（バッファ記憶）を提案し，グループＢに通常の記憶システムを提案した．



	Z (バイト)	τ (秒)	C (価格/バイト)
M_1	10^4	10^{-7}	10^2
M_2	10^6	10^{-6}	10^1
M′	10^6	τ'	C′
M_3	10^6	2×10^{-7}	2×10^1

試作モデルAには2つのカウンタ K_1 および K_2 が装備されている．K_1 は，M_1 において，“取出し”ができたとき計数され，K_2 は，M_2 によって“取出さ”なければならなかったとき計数される．1年間実地に使用した後で，$K_1/(K_1+K_2)=0.90$ であることがわかった．この結果，グループAはその2層記憶が M' なる通常の記憶システムと等価に見なせるといっている．

1. τ' の期待値はどれほどか．
2. C' の期待値はどれほどか．
3. 問1および2に答えるときどのような仮定を行なったか．
4. M' の仮定を考慮して，AよりもBの方がよいような状況が存在するか．存在すればそれを述べよ．
5. M' の仮定を考慮して，BよりもAの方がよいような状況が存在するか．存在すればそれを述べよ．

3.21

1. MULTICS はページングとセグメンテーションを行なっているシステムである．MULTICS のセグメント・マップ表はそれ自身セグメントであり，ディスクリプタ・セグメント(DS)と呼ばれる．DSを含め，

すべての MULTICS のセグメントはページに分けられている．ディスクリプタ・ベース・レジスタ（DBR）と呼ばれる特別のレジスタが DS のページ・マップ表の先頭を指している．MULTICS のようなシステムではどのような番地計算が行なわれるか．セグメント／オフセットの形式で指定される番地をアクセスするには，どのような手順を経なければならないか．本問では，MULTICS の詳細に触れようとするのではなく，ページングとセグメンテーションを行なうシステムにおける番地指定のためのハードウェアあるいはソフトウェアのオーバヘッドについて考察しようとするものである．

2. MULTICS システムでは，同時に幾人かの利用者の DS を主記憶に収容することが可能で，1人の利用者プログラムを実行するときには，適当な DS の先頭番地を DBR に設定する．現在の組込みにおいては，利用者を切換える手続き（SWAP_DBR）は番地変換の環境で動作し，すべての利用者の番地空間の中で同一のセグメント番号を持っている（すなわち，この手続きの主記憶番地を格納している DS のオフセットはすべての利用者に対して同一である）．したがって，DBR の内容を変更する命令を1つ実行することによって利用者を切換えることが可能である．この切換え手続き（SWAP_DBR）が各利用者ごとに異なったセグメント番号を持っていたとしたら，どのようなことが起こるか考察せよ．

3.22 デマンド・ページングとデマンド・セグメンテーションの2つは，利用者に大きな仮想記憶を与えることのできる記憶管理技法である：

1. どのようなときセグメント・スワッピングの方が優れているか述べよ．
2. 1つが他よりも優れている場合と，その理由を述べよ．

3.23 セグメンテーションのハードウェアがついている計算機があるとして，デマント・ページングを組込むことが要求されている．このためにはソフトウェアの技法だけを使用することができ，ハードウェアの追加を行なうことは認められない：

1. どのようにすればよいか詳細に述べよ．セグメンテーションのハードウェアについて考察し，ページングにどのような機能が必要かを考えよ．
2. 逆のことは可能か．すなわち，ページングのハードウェアがついてい

る機械が与えられて，セグメンテーションのシステムを組込むことは可能か．可能ならば簡単にその方法を述べよ．不可能ならばその理由を示せ．

3.24 たいていの計算機では，CPU は仮想記憶の番地指定を行なっても，入出力処理装置は“実際の”絶対番地が必要である．すなわち，オペレーティング・システムは，利用者の番地空間におけるチャネル・プログラムを入出力処理装置が処理可能な“実際の”チャネル・プログラムに変換する必要がある：

1. デマンド・ページングが行なわれていると仮定して，チャネル・プログラムを変換するに先立ってどのような動作と判定が必要か．
2. 記憶ブロックの状態として追加しなければならないのは何か．
3. ページの境界にまたがる入出力バッファを処理するにはチャネル・コマンド語のデータ・チェイニングをいかに使用すればよいか．
4. 変換アルゴリズムの流れ図を描け．
5. セグメンテーションが行なわれている場合には，この変換アルゴリズムはもっと簡単になるか，あるいは，もっと複雑になるか．理由と共に述べよ．

第4章

処理装置管理

処理装置管理は物理的な処理装置の管理，特に，プロセスへの処理装置の割付けを行なう．図 4.1 には次のモジュールが示されている：

ジョブ・スケジューラ　プロセスを作り，多重プログラミングを行なっている場合には，処理装置を与えるべきプロセスを決定する．

処理装置スケジューラ　多重プログラミングを行なっている場合に，どの実行可プロセスに，いつ，どれだけの期間，処理装置を与えるかを決定する．

トラフィック制御　プロセスの状態を把握する．

システム中のプロセスにはすべて同様の状態図が存在する．たいていのシステムでは，ジョブとプロセスの同期をとる必要が生じる．これもまた処理装置管理モジュールで行なわれる．

本章では次の3つの話題について論じる：

ジョブのスケジュール　(Abell, 1970；Browne, 1972；Coffman, 1968)

プロセスのスケジュール――トラフィック制御を含む．(Basket, 1970；Hansen, 1971；Kleinrock, 1970；Lampson, 1968；Oppenheimer, 1967；Varney, 1971；Buzen, 1971)

ジョブとプロセスの同期 (Bernstein, 1969；Dijkstra, 1968；Haberman, 1971；Horning, 1973；Spier, 1969)

図 4.1 はジョブ・スケジューリングとプロセス・スケジューリングの領分を示している．ジョブ・スケジューラは，どのジョブを実行させるかを選択する大局的なスケジューラと見なせる．プロセス・スケジューラは，スケジュールされたジョブに関連するプロセスに処理装置を割当てる局所的なスケジューラと見なせる．

利用者は，計算機システムで遂行したい仕事の集合がジョブであると見なし

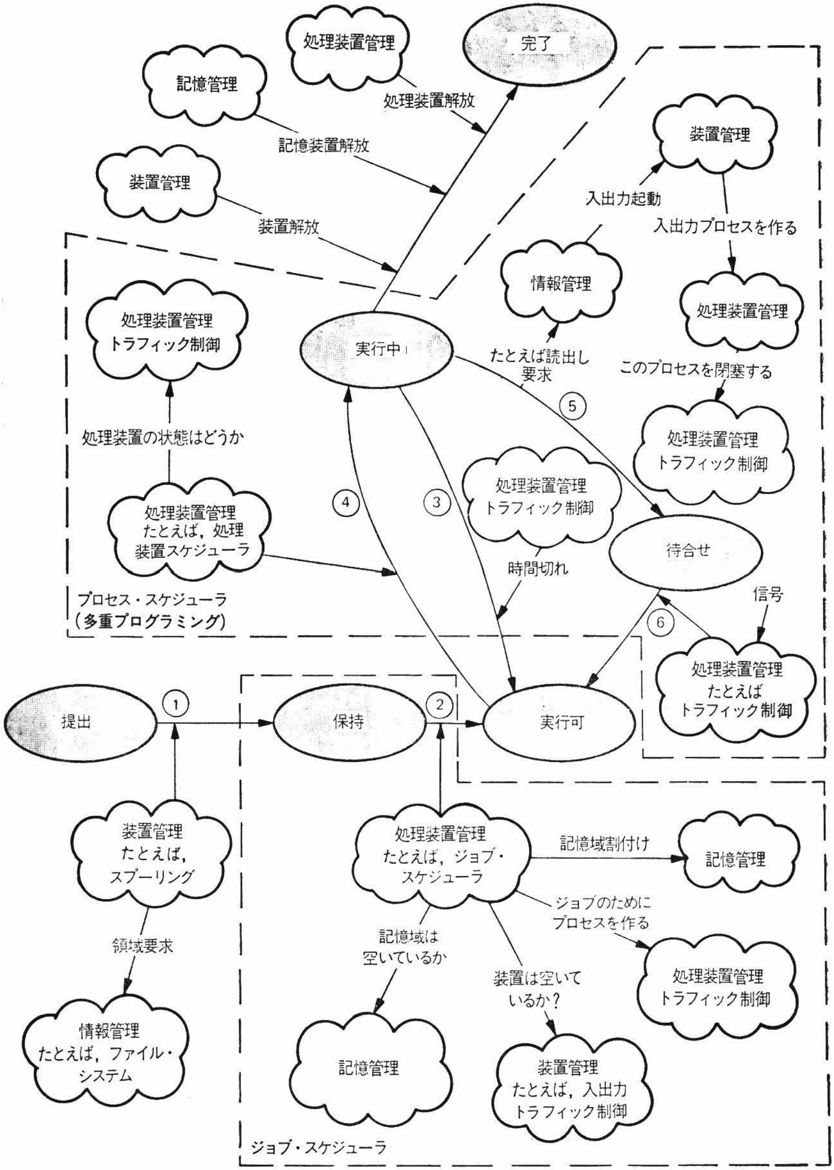


図 4.1　プロセスの観点から見たオペレーティング・システム

ている．利用者はジョブをジョブ・ステップに分割することができる．ジョブ・ステップは順番に処理される仕事（たとえば，コンパイル，ロード，実行）である．システムはジョブ・ステップの計算を行なうプロセスを作り出す．

ジョブ・スケジューリングはジョブの管理に関係する．処理装置のスケジューリングはプロセスの管理に関係する．どちらの場合にも，処理装置は基本的な資源である．

多重プログラミングを行なっていない場合には，ジョブとプロセスの間には1対1の対応が存在し，システムには，一時に1つのジョブだけが認められるので，プロセス・スケジューリングとジョブ・スケジューリングを区別していない（その必要もない）．このような簡単なシステムにおいてはジョブ・スケジューラは，実行のために1つのジョブを選択する．ジョブが選択されるとプロセスが作り出され，処理装置が割付けられる．

もっと一般的な多重プログラミング・システムでは，**ジョブ・スケジューラ**はシステムに提出されたジョブの中から一部を選び，システムに“入れる”．すなわち，ジョブ・スケジューラはこれらのジョブに対してプロセスを作り，プロセスに資源を割付ける．ジョブ・スケジューラは，たとえば，提出されている100のジョブからどのジョブを選んで資源を割付けたらよいかを決定する．**プロセス・スケジューラ**は，1群のプロセスのどのプロセスに，いつ，どれだけの期間，処理装置を与えたらよいかを決定する．

4.1 状態のモデル

1つのジョブについて図4.1の状態をたどってみよう．

プログラムは数日前に書かれたもので，適当な制御カードをつけて計算センタに提出された（**提出状態**）．オペレータはプログラムを計算機にかけて読込ませ，ジョブ・スケジューラが内容を調べるためのディスクに複写する（**保持状態**）．最後に，ジョブ・スケジューラはこのジョブを実行するために選択し，プロセスを生成する（**実行可状態**）．人手でスケジュールするシステムでは，より早くプロセスを実行可とするために，プログラマはカード・デックの前に緑色のカード（5ドル札）を置くこともある．ジョブ・スケジューラ・モジュールによって，実行のための処理装置を割当てられた（**実行状態**）．このとき，実行の時間の限度として“割当て時間”が与えられた．この割当て時間が経ったとき，計算は完了していなかった．したがって，プロセスは実行可状態に戻された．

しばらく経って，プロセス・スケジューラはこのプロセスに別の割当て時間を与え，処理装置を割付けた（実行状態）．この割当て時間にプロセスはほとんど完了したが答えの印刷が必要であった．したがって，このプロセスは別のプロセスとして入出力プロセスを起動し，入出力が完了するまで待合せなければならなかった（**閉塞状態**）．

入出力プロセスも同様な状態図を有している．このプロセスは CPU プロセスが入出力を起動すると**実行可**となった．入出力プロセスが完了すると，閉塞中であったプロセスに信号を送った．

入出力プロセスが完了したという信号をオペレーティング・システムが受取ると，トラフィック制御モジュールは CPU プロセスを閉塞状態から再び実行可状態にした．プロセス・スケジューラがこのプロセスに処理装置を割当てたとき（実行状態），すべての仕事は終了した（**完了状態**）．

処理装置管理には次のモジュールが含まれる：

モジュール	機能
スプーラ	1. 提出されたすべてのジョブをジョブ・スケジューラが処理できる形にする（詳細については第5章参照）
	提出 → 保持
ジョブ・スケジューラ	2. システム内のすべてのジョブを管理する 3. 実行すべきジョブを選択し，対応したプロセスを作る
	保持 → 実行可
プロセス・スケジューラ	4. 実行すべきプロセスを選択し，処理装置を割付ける
	実行可 ⇄ 実行中
トラフィック制御	5. すべてのプロセスの状態を管理する 6. プロセスの状態を変化する機構を用意する 7. プロセス間の同期と通信を調整する

4.1.1 ジョブ・スケジューラ

図4.1では，ジョブ・スケジューリングについて述べて，オペレーティング・システムにおけるプロセスの見方を示した．ジョブ・スケジューラは次のことを行なう“上位の”管理者である：

1. すべてのジョブの状態を把握する．どのジョブがサービスを受けようとしているか(保持状態)，および，サービス中のすべてのジョブの状態（実行可，実行中，閉塞中）はどうなっているかを知らせる．
2. どのジョブを“システムに入れるか”の方針を決定する（保持状態から実行可状態へ)．この決定は優先順位，要求された資源，あるいはシステムの負荷のバランスなどの特性に基づいて行なわれる．
3. 記憶管理，装置管理，および処理装置管理によって，スケジュールされたジョブに必要な資源を割付ける．
4. ジョブが終了したときこれらの資源を解放する．

4.1.2 プロセス・スケジューリング

ジョブ・スケジューラが1つのジョブを**保持状態**から**実行可状態**にするとき，このジョブのために1つまたはそれ以上のプロセスを作り出す．システムの中のどのジョブがいつ，どれだけの間，処理装置を与えられるかは何が決定するのであろうか．プロセス・スケジュールのためのモジュールがこの決定を行なう．特に，プロセス・スケジューラは次の機能を果たさなければならない：

1. プロセスの状態を把握する（プロセスの状態は実行中か，実行可か，閉塞中かである)．この機能を果たすモジュールは**トラフィック制御**と呼ばれる（Saltzer, 1960)．
2. どのプロセスがどれだけの間，処理装置を使用するかを決定する．これは**処理装置スケジューラ**によって行なわれる．
3. プロセスへ処理装置を割付ける．そのためには，処理装置のレジスタをこのプロセスの状態に正しく設定し直す必要がある．この仕事はトラフィック制御によつて行なわれる．
4. プロセスが割当て時間を使い果たしたときや入出力の完了を待合せなければならないときなどには，処理装置を解放する．そのためには，将来処理装置を割当てることができるように，処理装置のすべての状態レ

ジスタを退避しておく必要がある．この仕事はトラフィック制御によって行なわれる．

4.1.3 ジョブとプロセスの同期

ジョブの段階では，通常，ジョブ・ステップ間で条件を受渡す機構が用意されている．たとえば，もしジョブ・ステップ1が失敗したときにはジョブ・ステップ4の実行を中止したいことがある．

プロセスの段階では，競争条件を防止する機構が必要である．競争条件は，他のプロセスとの時間関係によつて計算結果が異なるときに起こる．たとえば，1つのプロセスが印刷している間に別のプロセスが印刷機を要求したとする．もしこの印刷機が第2のプロセスにも同時に割当てられるならば，印刷出力には2つのプロセスのものが混ざってしまうであろう．したがって，第1のプロセスが印刷機を解放するまで第2のプロセスの実行を停止させる機構が必要である．

PおよびVオペレータとセマフォアは，プロセスへの処理装置割付けを調整する機構である（Dijkstra, 1968)．さらに注意しておかなければならないのは，2つのプロセスが互に他が所有して手離さない資源を待合せて，すくみ（手詰まり）の状態になってしまうことである．このような場合には，これらのプロセスに処理装置を割当てることができない．

4.1.4 処理装置管理の構造

本章の主要な3つの節――ジョブ・スケジューリング，プロセス・スケジューリング，および同期――はプロセスへの処理装置の割当てを取扱っている．したがって，これらは処理装置について述べている本章に含められている．場合によっては，特にジョブ・スケジューリングの場合，他の資源管理と関係することがある．

要約すると，処理装置管理は，ジョブに処理装置を割付けることとプロセスに処理装置を割付けることの2つの段階で動作する．

ジョブの段階では，処理装置管理は，どのジョブが実行されることになるか，どのジョブが最初に実行されるかという問題に関係する．この第1段階では，処理装置管理は多重プログラミングとは関係がない．1つのジョブがスケジュールされたら，それは実行されるものと仮定している．ジョブ・スケジューリングは，他の利用者のプログラムやシステム中の他の処理と同時に実行することができる．したがって，ジョブ・スケジューリングは，図 4.2 に示し

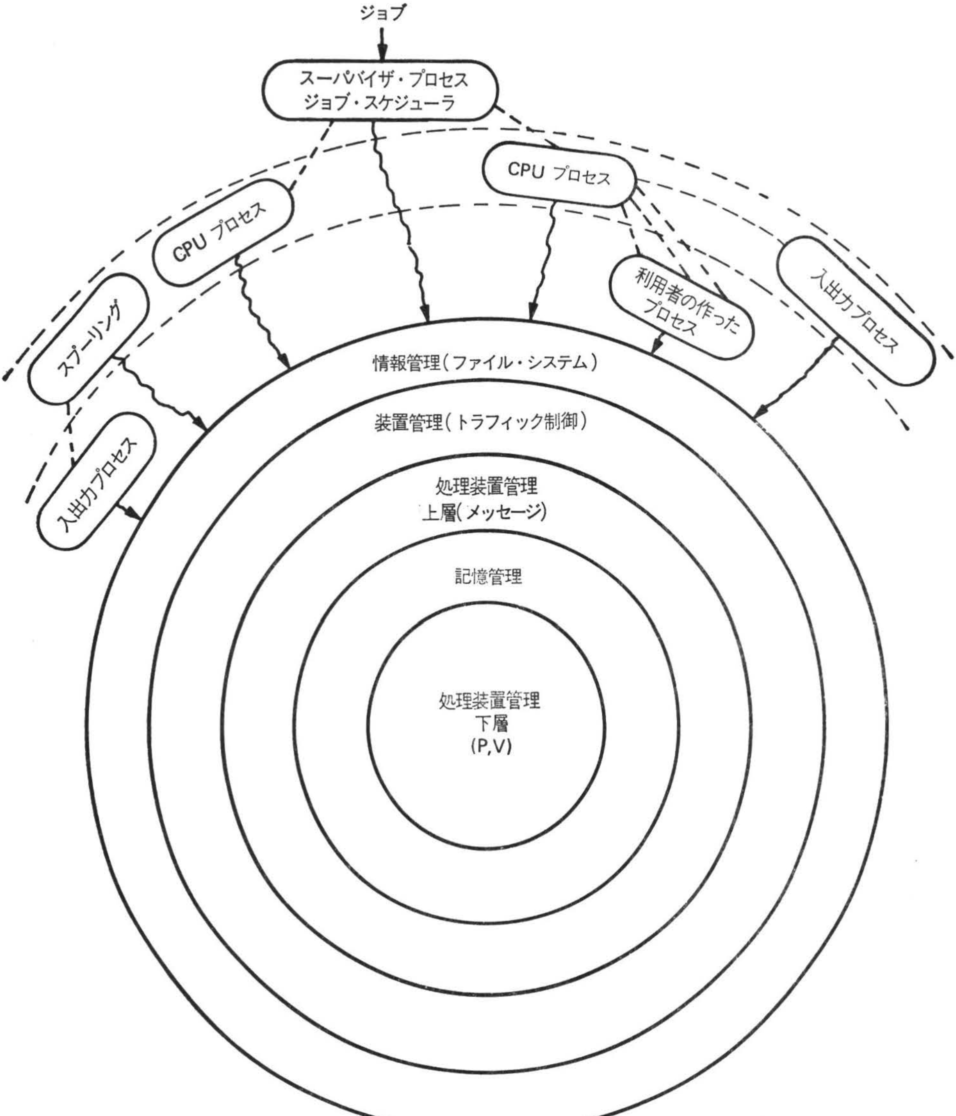


図 4.2 実の計算機あるいは拡張計算機に対するオペレーティング・システムの階層的な観点

たように，それ自体別個のプロセスである．

ジョブがスケジュールされると，システムは，プロセスを作り出したり，プロセスを破壊したり，プロセス間で通信したりする機能を果たさなければならない．一般的なシステムにおいては，これらの機能は利用者のプロセス，ジョブ・スケジューラ，あるいはその他のプロセスによって要求される．したがって，この機能はすべての番地空間に共通である（図 4.2――処理装置管理の上層――を参照）．

多重プログラミングを行なっている場合，プロセス・スケジューラと同期の機構は，システムのすべてのモジュールから呼出される．したがって．これらは，システムの核部分の中心を形成する（図 4.2――処理装置管理の下層――参照）．第7章の例題のオペレーティング・システムは，このシステム機構のモデルに基づいている．

ジョブ・スケジューラは競技会の役員のようなものである．この役員は，競技会に誰を出場させるかは決定するが，誰が勝つかについては知らない．プロセス・スケジューラは出場者の内の誰が勝つかを決定する．多重プログラミングを行なわない場合には，ただ1人の選手だけが出場を認められる．

4.2 ジョブ・スケジューリング

ジョブ・スケジューラの機能は，きわめて単純にも，きわめて複雑にも，あるいは動的にも行なうことができる．

コンパティブル・タイムシェアリング・システム（CTSS）（Crisman, 1965）などでは，ジョブ・スケジューリングの方針は，ログイン（すなわち，到着）している利用者を先着順に30人認めるというものであった．簡単な2段階の優先順位を認めるアルゴリズムでは，優先順位の高い方の利用者は順位の低い方の利用者を強制的にログアウト（すなわち，終了）させることができる．ログアウトされる利用者は，処理装置の使用時間を基準にして選択される．最も処理装置の使用時間の長い利用者がログアウトされる．

OS/VS2のようなバッチ・システムのジョブ・スケジューリング・アルゴリズムは複雑なものといえよう．このシステムでは，ジョブの到着時刻のみならず，優先順位，必要記憶容量，必要な装置の使用時間，およびシステムのバランスなども考慮される．

本節では，簡単なジョブ・スケジュールの方針を論じる．（第9章では，別のシステムの複雑なジョブ・スケジューラについて論じる．）ジョブ・スケジュ

ールの方針とそれらの関連に関して，最初は多重プログラミングを行なわない場合を，次にもっと一般的な多重プログラミング・システムの場合について，述べる．

4.2.1 機　　能

ジョブ・スケジューラは，ジョブにシステム資源を割当てる上級のスーパバイザと見なすことができる．（したがって，ジョブ・スケジューラは，ジョブの状態を把握し，どのジョブに資源を与えるかの方針を決定し，資源を割付けたり解放したりする．

本節では，ジョブ・スケジューリングの機能を果たす機構を微細に述べることはしない．興味のある読者は1つのスケジューリングの方法について，第7および第9章でコーディングのレベルまででも，希望する詳しさまで読んでほしい．本節では主としてスケジューリングを必要とする要件と，その結果について調べる．たとえば，ジョブ・スケジュールが記憶参照の必要性だけで行なわれたとすると，他のスケジュールの方針に比べてジョブのターンアラウンド時間にどんな影響が生じるか．

別の方針を論じる前に，システムの中でジョブの状態を把握する機能について簡単に述べよう．ジョブの状態を把握する1つの方法は，ジョブごとにジョブ制御ブロック（JCB）を持つことである．プロセスが保持状態にされたとき（図 4.1），JCB が作られ，これにプロセスの状態とジョブの行列の中での位置とが記入される．

図 4.3 には JCB に格納される情報の種類を示している．優先順位や予想処理時間などの情報は，ジョブと一緒に提出されるジョブ制御カードから求められる．状態などの他の情報はオペレーティング・システムによって設定される（JCB の詳細については第5.6.3.2節を参照）．

ジョブ識別名
状態
優先順位
予想時間
その他

図 4.3 ジョブ制御ブロック

4.2.2 方　　針

ジョブ・スケジューラは“保持”状態のジョブの中から適当なジョブを選んで，実行のために“実行可”状態としなければならない．小さな計算機センタでは，この仕事はオペレータによって行なわれる．オペレータは任意に，友人のジョブや最も処理時間の短いものを選ぶかもしれない．IBM OS/360 などのような大規模なシステムでは，提出されたジョブはすべて，いったん，2次記憶装置に格納される．ジョブ・スケジューラは2次記憶装置に格納したジョブを検査することができ，利用可能なシステム資源を使用するジョブを選び実行できるようにする．

方針を決定するに当たって，留意すべき点は次のとおりである：

基本的には，実行を要求しているジョブ数の方がシステムの資源で効率よく実行できるジョブ数よりも多い．特に，

a．たいていの資源は有限である（たとえば，磁気テープ，記憶装置）．

b．共有できない資源や他のプロセスに割付けし直すのが容易でない資源も多い．

スケジュールは，その目標が通常主観的な（しかも，しばしば矛盾している）ものであるから，方針決定の問題であると考えられる．たとえば，次の目標の重要性をどのようにして位づけするのか：

1. 1日当たりのジョブの処理件数を大きくする．（短時間ジョブのみを選ぶ．）
2. 処理装置の稼動率を大きくする．（計算を長時間行なうジョブのみを選ぶ．）
3. すべてのジョブを公平に扱う．（“公平”とは何か．）

ジョブ・スケジューリングの方針を決定するうえで考慮せねばならない点は次のとおりであろう：

1. 限定された特殊な資源の利用価値（たとえば，磁気テープ）

a．その資源に先取権を与えるならば，“悪用”されることがある．たとえば，プロッタを要求するジョブを先に実行するとすれば，つねにプロッタを要求する人が出るであろう（プロッタを使用しなくても）．

b．その資源に先取権を与えないとしたら，利用者によっては余分に待たされることがある（たとえば，スケジュールされるのを待ったうえ

で，磁気テープを装着したり，プロッタを調節したりするのを待たねばならない．

2. 急行のサービスには高い料金
3. システムの利用——処理装置時間と記憶容量——多くを要するほど長く待合せねばならない．
4. システムのバランス——入出力の多いジョブと CPU での計算の多いジョブを混在させる（たとえば，MIT ではジョブの型を A, B, C と区別している）．
5. サービスの保証——特定の待ち時間（1時間）の限度の設定をするか，一般的な限度（24時間以内）によるか（最低限，ジョブが紛失しないこと）．
6. そのジョブを定められた時間までに，または定められた時間に完了すること（たとえば，午後5時）．

ジョブ・スケジューラが実行すべきジョブを選択した後に，プロセス・スケジューラは微細なスケジュールを行なう（プロセスに処理装置を動的に割当てる）．割当て時間のことを無視しても，プロセスは，通常，100ミリ秒以内に入出力完了やその他の事象を待合せて閉塞状態となる．

ジョブ・スケジューラとプロセス・スケジューラは相互に関係して動作する．プロセス・スケジューラは，プロセスを選んで“遅延”させ（“ロールアウト”），ジョブ・スケジューラのスケジュールの下に戻すことがある．ジョブ・スケジューラは，このジョブを完了させるために，再度ジョブ段階でのスケジューリングを行なう．この例は第4.6節で論じるタイム・シェアリング・システムで見ることができる．上述の議論をすべて微細にわたって行なうことは不可能である．次節では2，3の簡単な例を示す．

4.2.3 多重プログラミングを行なわない場合のジョブ・スケジューリング

本節では多重プログラミングが行なわれないものと仮定する．すなわち，いったん1つのプロセスに処理装置が割当てられたら，プロセスが完了するまで処理装置を解放しない．本節ではまず**平均返却時間**を短くしようとする方針によって，ジョブをスケジュールする簡単な例を調べてみよう．多重プログラミングは行なわれていないので，各ジョブごとに1つの CPU プロセスが作り出されるものと仮定する．したがって，ジョブとプロセスは同じ意味を持つ．

4.2.3.1　FIFO によるジョブ・スケジューリング

図 4.4 に示すようにジョブが到着するものとする．

ジョブ番号	到着時刻	実行時間
1	10.00	2.00 時間
2	10.10	1.00 時間
3	10.25	0.25 時間

図 4.4　例題ジョブの到着時刻

ジョブ1は午前10時に到着し，ジョブ制御カードの推定処理時間は2時間である．本節ではジョブ処理時間は推定処理時間に等しいと仮定しておく．

FIFO アルゴリズムを用いると，ジョブは 図 4.5 に示すように実行される．平均返却時間は次式にて計算される：

ジョブ番号	到着時刻	開始時刻	終了時刻	返却時間
1	10.00	10.00	12.00	2.00 時間
2	10.10	12.00	13.00	2.90 時間
3	10.25	13.00	13.25	3.00 時間
				7.90 時間
平均返却時間 = 2.63 時間				

図 4.5 FIFO　によるジョブ・スケジューリング

$$T=(\sum_{i}^{n} T_i) * \frac{1}{n}$$

ただし，

$T_i=F_i-A_i$

F_i＝終了時刻

A_i＝到着時刻

4.2.3.2　最短時間のジョブから実行するジョブ・スケジューリング

平均返却時間を短縮する別のスケジューリング・アルゴリズムが存在するであろうか．図 4.6 は，最短処理時間の“保持”状態ジョブを先に実行するスケ

ジョブ番号	到着時刻	開始時刻	終了時刻	返却時間
1	10.00	10.00	12.00	2.00 時間
2	10.10	12.25	13.25	3.15 時間
3	10.25	12.00	12.25	2.00 時間
				7.15 時間
平均返却時間 = 2.38 時間				

図 4.6 最短ジョブを先に実行するジョブ・スケジューリング

ジューリング・アルゴリズムを用いる．すなわち，ジョブ1が到着したとき実行される．ジョブ1が実行中にジョブ2とジッブ3が到着する．ジョブ3の予想時間がジョブ2よりも短いので，次にはジョブ3が実行される．

このアルゴリズムでは平均返却時間が短縮されているが，“公平”だったであろうか．ジョブ2の利用者に尋ねてみるとよい．その利用者は公平だとは思っていない．

4.2.3.3 未来の知識を用いるジョブ・スケジューリング

平均返却時間をさらに改善する方法がほかにも存在するであろうか．たとえば，未来の知識を用いることは有用であろうか．午前10時に，間もなく短時間ジョブが2つ到着することがわかったとしたら，ジョブ1を実行したであろうか．図 4.7 は“未来の知識”を持つスケジューリング・アルゴリズムを使用した結果を示している．

平均返却時間は短縮したが，CPU 時間を0.25時間“無駄”にしたこと，およびジョブ1は運の悪い利用者になってしまったことに注意しなければならな

ジョブ番号	到着時刻	開始時刻	終了時刻	返却時間
1	10.00	11.50	13.50	3.50 時間
2	10.10	10.50	11.50	1.40 時間
3	10.25	10.25	10.50	0.25 時間
				5.15 時間
CPU アイドル時間 = .25 時間				
平均返却時間 = 1.72 時間				

図 4.7 未来の知識を用いるジョブ・スケジューリング

い．CPU をアイドルにすることが返却時間の改善に役立っているという事実は，多くの計算センタで，普通，忙しい午後の時間に低順位の1時間ジョブを開始させるより CPU をアイドルのままにしておく理由を説明している．

このジョブ・スケジュールのアルゴリズムは幾つかの問題点を有している：

1. 長時間ジョブを実行しないアルゴリズムを用いている1つのシステムが5年間動作して運転を終了した直後に，3年以上も紛失していたジョブが見つかったということになるかもしれない．
2. 未来の知識があることは滅多にない（もしあれば，スケジューリングなんかに使用せずに，株式市場でそれを用いればよい）．
3. 実行時間は近似値として予想される．
4. 記憶域の要求や入出力装置の要求などのその他の資源を考慮しなければならない．

上の問題点にもかかわらず，図4.5から図4.7の例は，良好なジョブ・スケジューラを用いると利益があることを示している（平均返却時間2.63時間が1.72時間に短縮されている）．1日に1000ジョブもを実行する大規模なシステムでは，常時，多数の“保持状態”のジョブが入っている．

4.2.3.4　スケジューリングの性能の尺度

前述の例では，スケジューリングの性能の尺度として返却時間を用いた．これ以外にも多くの尺度がある．たとえば，1分かかるジョブがスケジュールされて直ちに実行されると返却時間は1分である．一方，1時間のジョブがスケジュールされて直ちに実行されると返却時間は1時間となる．このことから1時間のジョブの処理が悪いといえるであろうか．否，実行時間よりも短い返却時間を与えることは不可能である．このような違いを正規化するために，**重みつき返却時間** $W=T/R$ という別の尺度を定義することができる．ここでTは前に計算した返却時間，Rは実行時間である．

図4.5，図4.6，図4.7の例では重みつき返却時間は，それぞれ 5.3，4.05，1.37時間である．これらの場合には，返却時間を改善した方針は同時に重みつき返却時間も改善している．このようなことはつねに成立するであろうか．次節に示す反例がこれを否定している．これらの理由によって，“このスケジューリング・アルゴリズムはよいか”という問に答える以前に性能の尺度を定義することが重要である．

4.2.4 多重プログラミングを行なう場合のジョブ・スケジューリング

前節では単純なジョブ・スケジューラの方針と技法を論じた．特に，平均返却時間Tを最小にするにはつねに最短処理時間のジョブを実行すればよい．

簡単のために，前節では一時に1ジョブが実行されるとした．しかし，実際にたいていのシステムでは多重プログラミングを行なっている．多重プログラミングを行なっている場合のジョブ・スケジューラの機能は実行すべき複数のジョブを選択することである．

本節では単純なプロセス・スケジューリングのアルゴリズムを仮定して，ジョブ・スケジューラの議論に焦点を合せよう．第4.3節では他のプロセス・スケジューリングのアルゴリズムについて論じるが，現在のところ，システムの中の各プロセスが短い割当て時間の間だけ処理装置を割当てられる **“ラウンド・ロビン”** 方式のアルゴリズムであるとしておく．すなわち，同時にnジョブが多重プログラミングで実行されているときには，それらのジョブは等分に実行時間が与えられる．

4.2.4.1 多重プログラミングを行なっているが入出力の同時処理を行なわない場合のジョブ・スケジューリング

多重プログラミングを行なうとスケジューリングにどのような影響を与えるであろうか．図 4.8 はジョブの混合体の例を示している．

図 4.9 **a** および **b** は，多重プログラミングを行なわない場合の FIFO 方式のスケジューリング・アルゴリズムの結果を示している．図 4.8 a はジョブが主記憶に帯在する時間を図示している．

図 4.10 は入出力が行なわれないとして（すなわち，すべてのジョブは CPU だけを使用する），同じジョブの混合体を多重プログラミングしたときの解析結果である．例では短いタイム・スライス（割当て時間）を仮定している．すなわち，2つのジョブが主記憶に入っている場合，CPU は交互に2つのジョブ

ジョブ番号	到着時刻	実行時間
1	10.0	0.3 時間
2	10.2	0.5 時間
3	10.4	0.1 時間
4	10.5	0.4 時間
5	10.8	0.1 時間

図 4.8 ジョブの到着時刻および実行時間

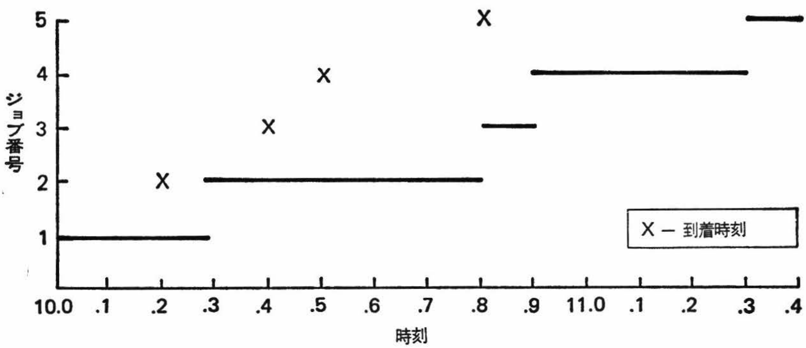


図 4.9 a FIFO――多重プログラミングなし（グラフ形式）

ジョブ番号	到着時刻	開始時刻	終了時刻	返却時間	重みつき返却時間
1	10.0	10.0	10.3	0.3 時間	1.00
2	10.2	10.3	10.8	0.6 時間	1.20
3	10.4	10.8	10.9	0.5 時間	5.00
4	10.5	10.9	11.3	0.8 時間	2.00
5	10.8	11.3	11.4	0.6 時間	6.00
				2.8 時間	15.20

平均返却時間 T = .56 時間
平均重みつき返却時間 W = 3.04

図 4.9 b FIFO――多重プログラミングなし（表形式）

を実行するから，2倍の時間，記憶装置に滞在することになる．

ここで **CPU の進行**という考えを用いることができる（関野，1972）．CPU の進行とは1つのジョブに使用された CPU 時間のことをいう．2つのジョブが多重プログラミングされている場合には，CPU の進行は時計が経過した時間の半分に等しい．この考えを用いて，多重プログラミングの場合，いつジョブが終了するかを計算してみよう．

図 4.10 の例では，主記憶容量と入出力装置には何ら制限がないものとする．したがって，ジョブは到着したら直ちに開始することができる．

ジョブ1が 10.00 AM に到着し，0.3 時間実行される予定であることに注意

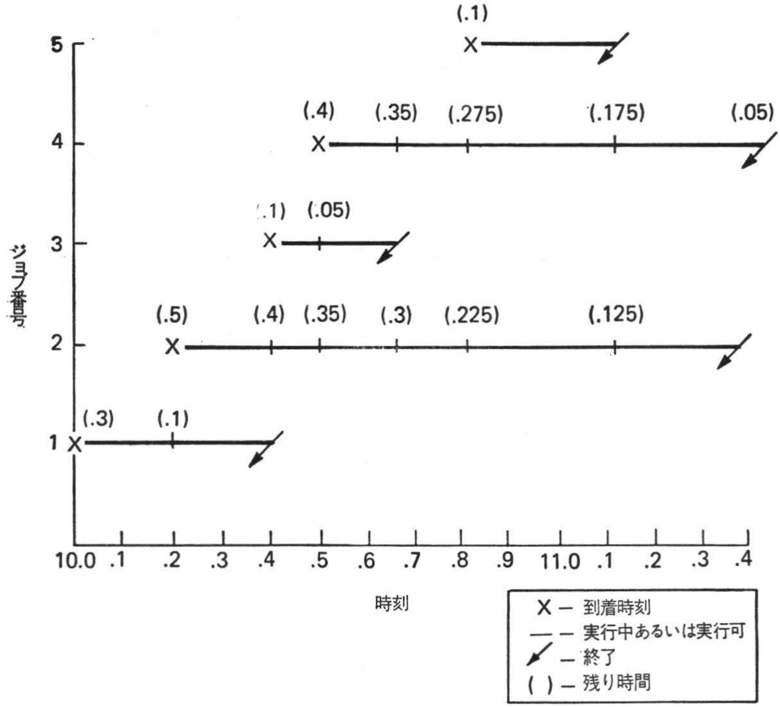


図 4.10a　多重プログラムされている5のジョブ——入出力同時実行なし

してほしい．ジョブ1が0.2時間実行したところでジョブ2が到着して主記憶に入れられている．時刻10.2から10.4の間では，処理装置はジョブ1とジョブ2の間で交互に切換えられている．したがって，ジョブ1の実行時間の残りはわずか0.1時間であるにもかかわらず，処理装置は2つのジョブにサービスをしているために，このジョブの終了にはあと0.2時間かかる．読者は図 4.10の他の時刻についても調べてみてほしい．

これからどんな結論が得られるであろうか．多重プログラミングは性能を改善するのかあるいは劣化させるのか．

図 4.9 および 図 4.10 からは，平均返却時間の点では，多重プログラミングは性能を劣化させていると結論できる．多重プログラミングを行なわないときの平均返却時間は0.56で，多重プログラミングのときは0.60である．（重みつき返却時間の平均値は改善されていることに注意．）

時刻	事象	ジョブ数	1ジョブ当りのCPU時間の率%	経過時間	1ジョブ当りの前進量	ジョブ	残り時間
10.0	ジョブ1到着					1	.3
10.2	ジョブ2到着	1	1	.2	.2	1 2	.1 .5
10.4	ジョブ3到着 ジョブ1終了	2	1/2	.2	.1	1 2 3	 .4 .1
10.5	ジョブ4到着	2	1/2	.1	.05	2 3 4	.35 .05 .4
10.65	ジョブ3終了	3	1/3	.15	.05	2 3 4	.3 .35
10.8	ジョブ5到着	2	1/2	.15	.075	2 4 5	.225 .275 .1
11.1	ジョブ5終了	3	1/3	.3	.1	2 4 5	.125 .175
11.35	ジョブ2終了	2	1/2	.25	.125	2 4	 .05
11.4	ジョブ4終了	1	1	.05	.05	4	

図 4.10 b　図 4.10 a の計算

ジョブ番号	実行時間	開始時刻	終了時刻	返却時間	重みつき返却時間
1	0.3	10.0	10.4	0.4 時間	1.33
2	0.5	10.2	11.35	1.15 時間	2.3
3	0.1	10.4	10.65	0.25 時間	2.5
4	0.4	10.5	11.4	0.9 時間	2.25
5	0.1	10.8	11.1	0.3 時間	3.0
				3.00 時間	11.38

平均返却時間 T = 0.6
平均重みつき返却時間 W = 2.276

図 4.10 c　FIFO——多重プログラミングを行なう

多重プログラミングが性能を劣化させるもっと明白な例も見出すことができる．図 4.11 では4つの1時間ジョブが同時に到着している．図 4.11 b ではすべてのジョブが多重プログラミングされている．平均返却時間と重みつき平均返却時間は共に4時間である．

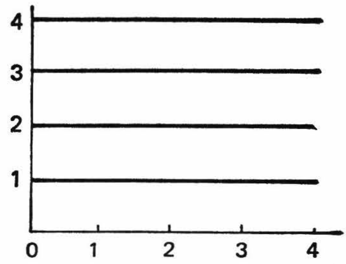
図 4.11 a では一時に1つのジョブが実行される．すなわち，1時間経つと1つのジョブが終了し，2時間目に第2のジョブ，3時間目に第3のジョブ，そして最後に，4時間目に第4のジョブが終了する．このとき平均返却時間と重みつき平均返却時間は2.5時間で，明らかに多重プログラミングより性能がよい．

多重プログラミングはつねに性能が悪いのであろうか．ジョブの混合のし方によっては否である．図 4.12 および図 4.13 を調べてみてほしい．

例のジョブが多重プログラミングされない場合（図 4.13）には3時間ジョブが最初に実行され，平均返却時間 T は 3.6 時間となる．多重プログラミングの場合には，図 4.12 に示したように，大幅に改善され，T はただの 1.88 時間である．

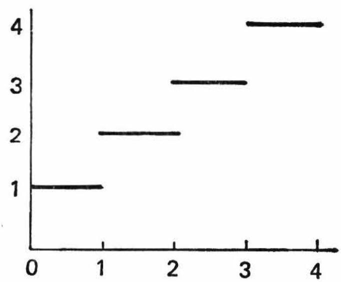
4.2.4.2 多重プログラミングと入出力の同時実行を行なう場合のジョブ・スケジューリング

実際には，ジョブの実行時間の一部は入出力**待合せ**時間であり，この時間は重複させることが可能である（すなわち，入出力が入出力チャネルで処理中はCPU を他の計算のために使用することが可能である．このようにすると多重



平均返却時間，T＝4時間
平均重みつき返却時間，W＝4

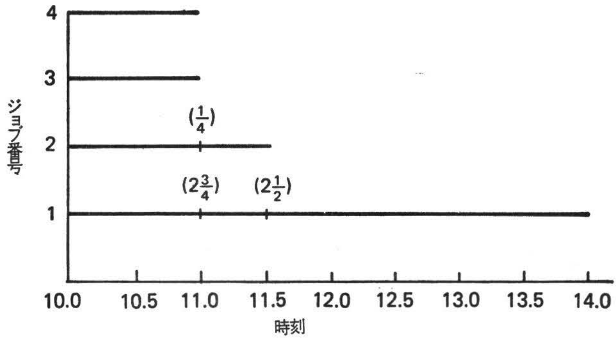
(a) 多重プログラミングを実行するとき



平均返却時間，T＝2.5時間
平均重みつき返却時間，W＝2.5

(b) 一時に1つのジョブを実行するとき

図 4.11　多重プログラミングが性能を悪くする例



(a) グラフ

ジョブ番号	実行時間	開始時刻	終了時刻	返却時間	重みつき返却時間
1	3.0	10.0	14.0	4.0 時間	1.3
2	0.5	10.0	11.5	1.5 時間	3.0
3	0.25	10.0	11.0	1.0 時間	4.0
4	0.25	10.0	11.0	1.0 時間	4.0
				7.5 時間	12.3

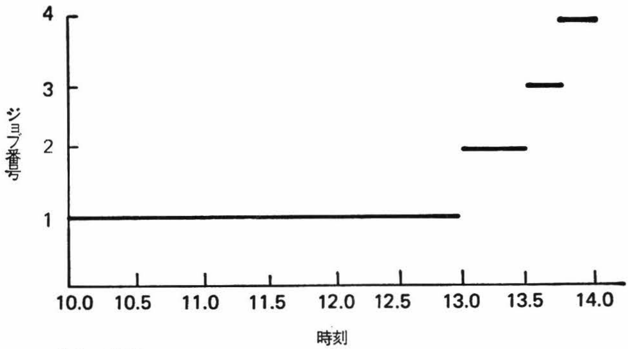
平均返却時間，T＝1.88時間
平均重みつき返却時間，W＝3.1

図 4.12　多重プログラミングの場合のスケジューリングの例

プログラミングの性能にどのように影響するであろうか．ジョブの時間の25%が待合せ時間であるとしよう．すなわち，単一プログラミングによって実行されたとしたら，経過時間の 25% の間 CPU は入出力を待合せている．

しかし，多重プログラミングを行なっている場合には，この待合せ時間の間，処理装置を別のジョブに割当てることが可能である．しかし，すべてのジョブが同時に入出力待合せを行なう可能性もあり，この場合には第 3.2 節で論じたように，CPU は部分的にアイドルとなる．システムの待合せ率の表を再び 図 4.14 に示す．この表の計算については第 4.3 節で述べている．

ジョブの平均返却時間を計算するためにはいつジョブが終了するかを決定し



(a) グラフ

ジョブ番号	実行時間	開始時刻	終了時刻	返却時間	重みつき返却時間
1	3.0	10.0	13.0	3.0 時間	1.0
2	0.5	10.0	13.5	3.5 時間	7.0
3	0.25	10.0	13.75	3.75 時間	15.0
4	0.25	10.0	14.0	4.0 時間	16.0
				14.25 時間	39.0

平均返却時間，T＝3.56時間
平均重みつき時間，W＝10.0

(b) 表

図 4.13 多重プログラミングを行なわない場合のスケジューリングの例

なければならない．このためには，CPU の進行，すなわち，一定の時間の間に1つのジョブがどれだけの CPU の“仕事”を受取るかを理解しておく必要がある．

2つの場合を考えよう．まず最初に1つのジョブが単独で実行されていて入出力待合せが25% とすると，1時間経過すると 0.75 時間の CPU の進行が行なわれる．したがって，1.5 時間の CPU 時間が必要なときは，このジョブが完了するのに2時間かかることになる．

次にもっと複雑な場合で，同時に開始し，両方とも25%の入出力待合せを行ない，1.5 時間の CPU 時間を必要とする2つのジョブの場合を考える（単独

プログラムの単独実行の場合の待合せ時間率

多重プログラミングの多重度

1	5.0	10.0	15.0	20.0	25.0	30.0	35.0	40.0	45.0	50.0	55.0	60.0	65.0	70.0	75.0	80.0	85.0	90.0	95.0
2	0.1	0.6	1.3	2.4	4.0	6.0	8.6	11.8	15.5	20.0	25.2	31.0	37.6	45.0	52.9	61.5	70.7	80.2	90.0
3	0.0	0.0	0.1	0.2	0.4	0.9	1.5	2.5	4.1	6.3	9.3	13.4	18.9	25.9	34.6	45.1	57.2	70.6	85.1
4	0.0	0.0	0.0	0.0	0.0	0.1	0.2	0.4	0.8	1.5	2.8	4.8	8.1	13.1	20.6	31.1	44.7	61.4	80.2
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.3	0.7	1.4	2.9	5.8	11.0	19.9	33.6	52.5	75.3
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.4	0.9	2.2	5.2	11.7	24.1	44.1	70.4
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.2	0.7	2.2	6.3	16.3	36.2	65.7
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.2	0.8	3.0	10.4	28.9	60.9
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.3	1.3	6.1	22.4	56.3
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.5	3.4	16.8	51.7
11	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	1.7	12.1	47.2
12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.8	8.3	42.7
13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	5.4	38.5
14	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	3.4	34.3
15	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	2.0	30.3
16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.1	26.4
17	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6	22.8
18	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	19.4
19	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	16.3
20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	13.4

図 4.14　表：多重プログラミングを行なった場合の待合せ時間率

で実行するとおのおの2時間かかる）．これらのジョブはいつ終了するであろうか．図 4.14 において2ジョブの場合には CPU がアイドルになるのは 4% の時間である．1時間の内で両方のジョブに対する CPU の前進の合計はどれだけになるか．それは 0.96 CPU 時間である．この量を2つのジョブに等分すると，各ジョブは1時間当たり 0.48 時間だけの CPU の前進を得る．したがって，2つのジョブを完了するには 1.5÷0.48＝3.13 時間かかる．これはジョブを単一プログラミングで順次処理した場合の4時間に比べて大幅な改善である．

図 4.15 は，図 4.12 に示したジョブが 25% の入出力待合せ時間を持っているとして計算した平均返却時間である．図には，図 4.10 で示したのと同様に，ジョブの到着と完了の事象が示されている．これらの事象と事象の間の時間には固定数のジョブが実行されている．図 4.15 b の“ジョブ数”の欄は，前回の事象以後この欄の事象が起こるまで同時実行されていたジョブの個数である．全体の CPU の待合せ時間は，各ジョブの待合せ時間が25%であると仮定して図 4.14 から求められる．“% CPU/ジョブ” の欄は，ジョブごとに使用される CPU 時間の割合である．この値に経過時間を乗算すると，ジョブごとの CPU の前進量が求められる．ジョブごとに前回の残り時間からこの前進時間を減じると，今回の残り時間が求められる．読者は是非とも図 4.15 の計算を自分で確かめてほしい．

返却時間に対する多重プログラミングの影響のために，この解析はしばしばきわめて複雑なものとなる．本節では，すべてのジョブは25%の入出力待合せ

ジョブ	プログラム実行時間	開始時刻	終了時刻	返却時間	重みつき返却時間
1	0.3	10.0	10.356	0.356 時間	1.187
2	0.5	10.2	11.05	0.850 時間	1.700
3	0.1	10.4	10.581	0.181 時間	1.810
4	0.4	10.5	11.158	0.658 時間	1.645
5	0.1	10.8	11.025	0.225 時間	2.250
				2.270 時間	8.592

平均返却時間，T＝.454時間
平均重みつき返却時間，W＝1.718

図 4.15 a　多重プログラミングと入出力の同時実行を行なう場合のジョブ・スケジューリング：要約

時間を有していると仮定している．実際のシステムでは，入出力待合せ時間の割合はジョブごとに異なっている．そのうえに，今日の計算機での入出力待合せの割合は普通50〜80%の範囲になっている．

時刻	事象	ジョブ数	CPU 待合せ率	1ジョブ当りの CPU時間の率%	経過時間	ジョブ	1ジョブ当りの前進量	必要とする CPU時間
10.0	ジョブ1 到着					1		.225
10.2	ジョブ2 到着	1	25	75	.2	1 2	.15	.075 .375
10.356	ジョブ1 終了	2	4	48	.156	1 2	.075 .075	 .3
10.4	ジョブ3 到着	1	25	75	.04375	2 3	.033	.267 .075
10.5	ジョブ4 到着	2	4	48	.1	2 3 4	.048 .048	.219 .027 .3
10.581	ジョブ3 終了	3	0	33.3	.081	2 3 4	.027 .027 .027	.192 .273
10.8	ジョブ5 到着	2	4	48	.219	2 4 5	.105 .105	.087 .168 .075
11.025	ジョブ5 終了	3	0	33.3	.225	2 4 5	.075 .075 .075	.012 .093
11.05	ジョブ2 終了	2	4	48	.025	2 4	.012 .012	 .081
11.158	ジョブ4 終了	1	25	75	.108	4	.081	

図 4.15 b　多重プログラミングと入出力の同時実行を行なう場合のジョブ・スケジューリング：表による解析

CPU の前進と返却時間の拡張概念は重要である．計算センタで採用されている進歩した計算料金計算法の多くは，ジョブの単独実行の場合の実行時間を求めようとしている．このようにすることによって，他のジョブが同時に多重プログラミングされていても，ジョブ実行の料金に変化がないようにすることができる（すなわち多重プログラミングを行なっている場合には，実際の経過時間は料金とは無関係である）．

4.2.4.3 記憶域の要求を行ない入出力の同時実行を行なわないジョブ・スケジューリング

CPU 時間に加えて，ジョブは一定の記憶域を必要とする．記憶容量に制限

ジョブ番号	提出時刻	実行時間(時間)	必要記憶容量(K)
1	10.0	0.3	10
2	10.2	0.5	60
3	10.4	0.1	50
4	10.5	0.4	10
5	10.8	0.1	30

図 4.16 記憶容量を考慮するスケジューリングでのジョブ例

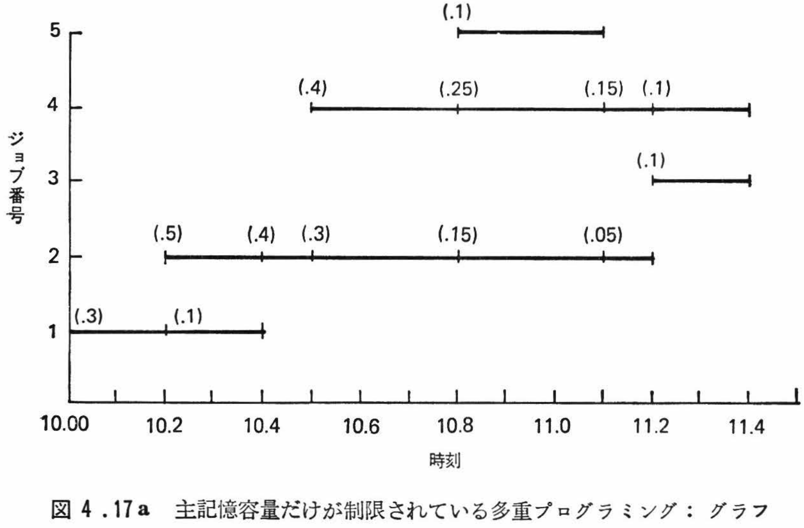


図 4 .17 a 主記憶容量だけが制限されている多重プログラミング：グラフ

時間	事象	ジョブ番号	1ジョブ当りのCPU時間率	経過時間	1ジョブ当りの前進量	空き記憶容量	ジョブ	残り時間
10.0	ジョブ1到着 ジョブ1開始					90	1	.3
10.2	ジョブ2到着 ジョブ2開始	1	1	.2	.2	30	1	.1
							2	.5
10.4	ジョブ1終了 ジョブ3到着	2	1/2	.2	.1	40	1	
							2	.4
10.5	ジョブ4到着 ジョブ4開始	1	1	.1	.1	30	2	.3
							4	.4
10.8	ジョブ5到着 ジョブ5開始	2	1/2	.3	.15	0	2	.15
							4	.25
							5	.1
11.1	ジョブ5終了	3	1/3	.3	.1	30	2	.05
							4	.15
							5	
11.2	ジョブ2終了 ジョブ3開始	2	1/2	.1	.05	40	2	
							3	.1
							4	.1
11.4	ジョブ3終了 ジョブ4終了	2	1/2	.2	.1	100	3	
							4	

図 4.17 b　記憶容量だけが制限されている多重プログラミング：表

ジョブ番号	実行時間	到着時刻	終了時刻	返却時間	重みつき返却時間
1	0.3	10.0	10.4	0.4 時間	1.33
2	0.5	10.2	11.2	1.0 時間	2.0
3	0.1	10.4	11.4	1.0 時間	10.0
4	0.4	10.5	11.4	0.9 時間	2.25
5	0.1	10.8	11.1	0.3 時間	3.0
				3.6 時間	18.58

平均返却時間，T = .72時間
平均重みつき返却時間，W = 3.716

図 4.17 c　記憶容量だけが制限されている多重プログラミング：要約

があり，再配置型割付け法（断片化を解消する方法）を用いるとき返却時間にどのような影響があるだろうか．

図 4.16 のように FIFO のジョブ・スケジューリングを用いた場合の解析結果が図 4.17 に示されている．記憶容量は 100K で，入出力の同時実行を行なわない多重プログラミングを行なっているものとしている．ジョブ 3 は時刻 10.4 に到着するが，50K の記憶域を必要とし，時刻 11.2 になるまでは記憶域が空かないので，この時刻まで実行が開始されない．

この記憶域の制限によって平均の待合せ時間が増加していることが図に示されている．

4.2.4.4　記憶容量と磁気テープに制限があり，入出力の同時実行を行なわないジョブ・スケジューリング

記憶容量と CPU 時間のほかに磁気テープ駆動装置を何台か要求するジョブを考えてみよう．この場合の返却時間に対する影響は何であろうか．

図 4.18 にはジョブが出す要求の例が示されている．システムの記憶容量は 100K で磁気テープ装置は 5 台設置されていると仮定する．

図 4.19 にはこの場合の解析結果が示されている．ジョブ 5 は十分な台数の磁気テープ装置が空くまでは開始できないことに注意してほしい．磁気テープ装置の制限を追加すると，図 4.17 に示した記憶域の制限だけの場合よりも，重みつきの平均返却時間が長くなっている．

ジョブ番号	提出時刻	実行時間 (時間)	必要記憶量(K)	磁気テープ必要台数
1	10.0	0.3	10	2
2	10.2	0.5	60	1
3	10.4	0.1	50	4
4	10.5	0.4	10	2
5	10.8	0.1	30	3

図 4.18 記憶容量と磁気テープ台数を考慮する
スケジューリングでのジョブ例

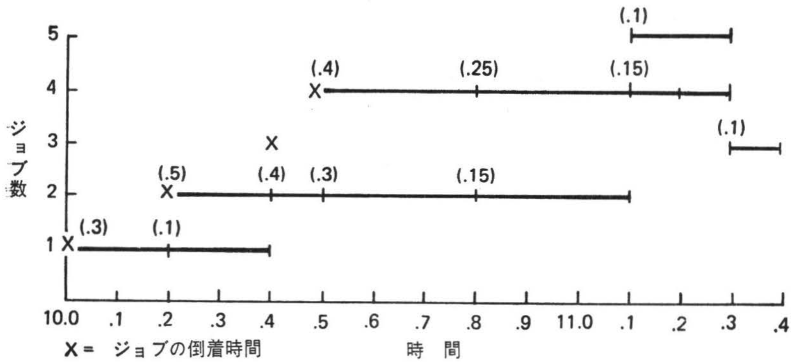


図 4.19 a 記憶容量と磁気テープ台数を考慮する多重プログラミング： グラフ

4.2.5 ジョブ・スケジューリングの要約

ジョブ・スケジューリングのアルゴリズムでは返却時間を最短にすることが一般的に行なわれる．しかし，実際のシステムではこの方針を一部変更する必要も生じる．その理由としては次のものがあげられる：

1. “優先順位”の考えによって特定のジョブを優先する——特別料金を支払っていることが多い．
2. 24時間以内に1日の仕事を完了できるような処理能力が必要である．（ある種の技法では CPU をアイドルにすることもあることを注意してほしい.）

時刻	事象	ジョブ番号	1ジョブ当りのCPU時間率	経過時間	1ジョブ当りの前進量	空き記憶容量	空き磁気テープの台数	ジョブ	残り時間
10.0	ジョブ1到着 ジョブ1開始					90	3	1	.3
10.2	ジョブ2到着 ジョブ2開始	1	1	.2	.2	30	2	1	.1
								2	.5
10.4	ジョブ1終了 ジョブ3到着	2	1/2	.2	.1	40	4	1	
								2	.4
10.5	ジョブ4到着 ジョブ4開始	1	1	.1	.1	30	2	2	.3
								4	.4
10.8	ジョブ5到着	2	1/2	.3	.15	30	2	2	.15
								4	.25
11.1	ジョブ2終了 ジョブ5開始	2	1/2	.3	.15	60	0	2	
								4	.1
								5	.1
11.3	ジョブ4終了 ジョブ5終了 ジョブ3開始	2	1/2	.2	.1	50	1	3	.1
								4	
								5	
11.4	ジョブ3終了	1	1	.1	.1	100	5	3	

図 4.19 b　記憶容量と磁気テープ台数を考慮する多重プログラミング：表

要約すると，設計者は平均返却時間を小さくするなどの目標をまず設定し，代表的なジョブの混合体に対して幾つかのスケジューリング・アルゴリズムを実験してみなければならない．

第 4.2 節においては数種類のスケジューリング・アルゴリズムについて調べ

ジョブ	実行時間	到着時刻	終了時刻	返却時間	重みつき返却時間
1	0.3	10.0	10.4	0.4 時間	1.33
2	0.5	10.2	11.1	0.9 時間	1.80
3	0.1	10.4	11.4	1.0 時間	10.00
4	0.4	10.5	11.3	0.8 時間	2.00
5	0.1	10.8	11.3	0.5 時間	5.00
				3.6 時間	20.13

平均返却時間，T = .72時間
重みつき返却時間，W = 4.026

図 4.19 c　記憶容量と磁気テープ台数を考慮する多重プログラミング：要約

てみた．最初の3つの方法では同一のジョブ混合体について適用してみた．この混合体に対する結論は次のとおりである：

1. FIFO 方式は各ジョブの待合せ時間の幅を小さくする．
2. 最短ジョブを先に処理する方式は平均の返却時間を小さくするが，あきらかに短いジョブを優遇している．
3. 未来の知識を用いる方式は最良の返却時間を与えるが，組込むのが困難である．

多重プログラミングを行なっている場合には，ジョブの混合のしかたによっては返却時間に差があることがわかった．多重プログラミングの場合には入出力の同時実行を行なうスケジューリングについてもっと複雑な解析をも行なった．記憶域と入出力装置の要求についても考慮した．他のジョブ・スケジューリング・アルゴリズムについては第9章で議論する．

4.3 プロセス・スケジューリング

第4.2節では図4.1の遷移2を行なうジョブ・スケジューリングについて論じた．プロセス・スケジューリングは遷移3および4に関連する．

ジョブ・スケジューラが処理のためにジョブを選択すると，プロセス・スケジューラは微細なスケジューリングを行なおうとする（すなわち，プロセスに処理装置を動的に割当てること）．プロセス・スケジューラは文献によっては**ディスパッチャ**とか**下層のスケジューラ**とも呼ばれている．多重プログラミング

を行なっている場合には，プロセスが処理装置を使用し始めてからわずか100ミリ秒以内に入出力待合せやその他の事象の待合せのために閉塞状態となることが普通である．

第4.2節の多重プログラミングの例では，処理装置が主記憶に入っているジョブに平均に分割して与えらるようにラウンド・ロビン方式の簡単なプロセス・スケジューリング・アルゴリズムを仮定していた．しかし，本節で述べる理由によって，もっと複雑なプロセス・スケジューリング・アルゴリズムが必要とされる．

ジョブ・スケジューラとプロセス・スケジューラとは互に関連して動作する．プロセス・スケジューラは，1つのプロセスの処理を中断してロールアウトし，ジョブ・スケジューラの段階での制御に戻すように要求することがある．このようなことはタイムシェアリング・システムではよく行なわれる．

図4.20には単一処理装置での多重プログラミングが示してある．プロセスAとDは共に実行の間に磁気テープ装置を要求している．使用できる磁気テープ装置は1台だけであり，プロセスAが先に要求を出したと仮定する．この装置はプロセスAに割当てられる．後刻プロセスDが磁気テープ装置を要求したときには，プロセスAが磁気テープ装置を解放するまで閉塞状態とならなければならない．

閉塞状態はプロセスの状態であり，処理装置の状態ではない．すなわち，処理装置は閉塞状態とはならなずA，B，Cのプロセス間で切換えられ，プロセスDを無視するだけである．通常，プロセスDは記憶域の一部を占有したままとなっているが，システムによっては主記憶から2次記憶へロールアウトしたりページアウトすることもある．このような場合には，プロセスDを閉塞状態とすることは資源の再分配に論理的な影響を及ぼす．資源は無駄に余らせておく必要はない．このことはプロセスの同期が必要となることから生じる処理装

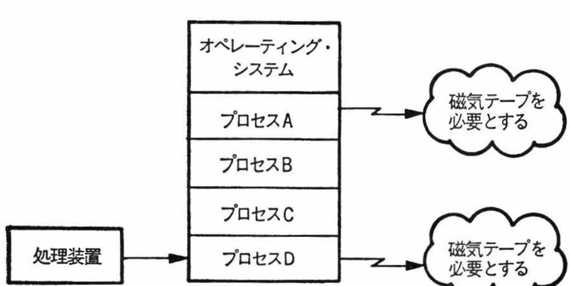


図4.20　単一処理装置のインターロック

置スケジューリングの問題点の一例である.

4.3.1 機　　能

プロセス・スケジューラは次の機能を果たさなければならない：

1. プロセスの状態を把握する.
2. どのプロセスが，いつ，どれだけの期間，処理装置を使用するかを決定する.
3. プロセスに処理装置を割付ける.
4. プロセスから処理装置を解放する.

前節と同様に本節でも，処理装置スケジューリングの組込み技法の詳細には立入らないことにする．これらの詳細は計算機に依存する部分が多く，概念的には重要なことではない．詳細について知りたい読者は，第7章の例題のオペレーティング・システムまたは第9章の説明を参照してほしい．本節ではスケジューリングの方針に焦点を合せて述べる.

プロセスの状態を把握する機能を簡単に論じよう．この機能を果たす処理装置スケジューラのモジュールはトラフィック制御と呼ばれる．プロセスの状態を把握するための1つの方法は，システム中の各プロセスに対してプロセス制御ブロック（PCB）と呼ばれるデータ•ベースを使用することである．PCBはプロセスごとに別個に存在する．PCBはプロセスにおける唯一の触れてみることのできる実体である（図4.21参照)．同一の状態（たとえば，実行可，閉塞など）をとっているすべてのプロセスのPCBは連結されていることが多く，**実行可リスト**などという名前がつけられたリストとなっている．トラフィック制御はプロセスの状態が変化するときに呼出される．**閉塞リスト**は，プロセス

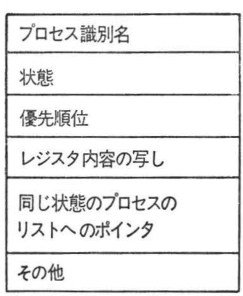


図 4.21 プロセス制御ブロック

が閉塞された理由によってさらに別々のリストに分割されることがある．したがって，プロセスがすでに使用中の装置を要求したときには，このプロセスはその装置に関連した閉塞リストに結合される．

その装置が解放されたときには，トラフィック制御はその閉塞リストをみて，その装置を待合せているプロセスがないかを調べる．待合せているプロセスが存在すれば，これらのプロセスは実行可の状態に戻され，再びその装置を要求する．別の方法としては，1つのプロセスにこの装置を割当てて実行可状態とし，残りのプロセスはそのまま閉塞状態にしておくこともある．

4.3.2 方　　針

スケジューリングとは，どのプロセスにどれだけの期間処理装置を割当てるかを決定することである．処理装置がプロセスに割当てられる時間の長さは次の事象の1つあるいはこれらの組合せによって決定される：

プロセスが完了する．
プロセスが閉塞状態となる．
優先順位の高いよりプロセスが処理装置を要求した．
割当て時間が経過した．
誤りが発生した．

どのプロセスを実行させるかに関しては，PCB の実行可リストを走査して一定の方針を適用して決定する．

実行可リストは2つの方法で構成することができる：

1. プロセスを（保持，実行，あるいは閉塞状態から）実行可リストに入れるときは優先順位にしたがって並べる．処理装置が空いたときには，リストの先頭にあるプロセスが実行される．
2. プロセスは実行可リストに任意の順で置かれる．実行すべきプロセスを見つける必要があるときは，スケジューラは実行可リスト全体を走査して実行すべきプロセスを選ぶ．

普通のプロセス・スケジューリングの方針には次のようなものが含まれている：

1. **ラウンド・ロビン**　各プロセスは，たとえば100ミリ秒の割当て時間がくるまで順番に実行される．
2. **残り時間の逆数**　プロセスが前回に割当て時間全部を使用していた

ならばリストの最後に置かれる．半分の時間しか使用しないときは（入出力待合せのために），リストの中央に置かれる．この方法は“公平”であると同時に，入出力の多いジョブに対しては良好に動作する．

3. **多段階のフィードバックのあるラウンド・ロビン**　プロセスが新しく作り出されたときは，このプロセスはシステム中の他のジョブがすでに使用しただけの時間をまず与えられて動作する．その後で正規のラウンド・ロビン方式となる．
4. **優先順位**　最も優先順位の高い実行可のジョブが選択される．優先順位は料金によって決定されたり，ジョブ種別によって決定される（たとえば，原子炉を制御するプロセスは第1に実行される）．
5. **限定されたラウンド・ロビン**　ジョブは一定の回数ラウンド・ロビンで実行される．その後は，システムに他のジョブが存在しないときだけ実行される．
6. **システム・バランス**　入出力装置を稼動状態に保つために，入出力の多いジョブは優先順位が高くされる．
7. **“会話型”ジョブを優先的に処理する**　利用者がプロセスと直接交信しているとき(会話型プロセス)，迅速なサービスを行なうために利用者の入力が終ると直ちに処理装置がプロセスに与えられる．
8. **ジョブの価値**　場合によってはシステム自身が優先順位を割当てることができると都合のよいことがある．たとえば，短時間のジョブに高い優先順位を与えてもよい．別の例としては，入出力要求の多いジョブと入出力要求の少ないジョブを同時に実行して平衡のとれた負荷となるように優先順位を定めることができる．すべてのジョブが一方に偏ってしまうと，入出力チャネルが過負荷となりCPUがアイドルになったり，あるいはその逆の現象が起こる．したがって，システムが適当にジョブを混合して平衡のとれた負荷を維持することは，システムの仕事量を最大にするために望ましいことである．同様にして，システムは資源を拘束しているジョブの優先順位を高め，ジョブが完了して資源を解放し，他のジョブが使用できるようにしたいこともある（以前にスケジュールされた磁気テープ装置など）．

別の処理装置スケジューリングの方針については文献で論じられている(Buzen, 1973 ; Coffman, 1968)．

たいていの配給の場合と同時に，何が公平で，効率的で，経済的な方策かという点に関しては議論のあるところである．読者はスケジューリングの方針の

詳細については参考文献を参照してほしい．この分野では多数の計算機民俗学が発展している．

4.3.3 スケジューリングのためのプロセス状態図

図 4.22 に示したようなプロセス状態図を用いると，プロセス・スケジューリングのアルゴリズムを説明するのに都合がよい．この例では，2つの実行可状態が示されている(低順位実行可と高順位実行可)．プロセスは，実行中に割当て時間を経過したとき低順位実行可状態となり，閉塞状態から実行可状態となったとき高順位実行可状態となる．

スケジューラの方針は次のようになるであろう：

1. 高順位実行可リストから実行するプロセスを選択する．
2. 高順位実行可リストにプロセスが存在しないときには，低順位実行可リストからプロセスを選択する．

図 4.22 の状態遷移は，プロセスを入出力限界（高順位実行可）(bound) または CPU 限界（低順位実行可）のいずれかに特色づける効果がある．このスケジューラの方針は入出力型のプロセスを優先的に処理する．

もっと複雑なスケジューリング・アルゴリズムは図 4.23 に示されている．このスケジューリング・アルゴリズムは，ページングによる記憶管理を行なうタイムシェアリング用のオペレーティグン・システムで用いることができる．特に，閉塞状態のプロセスは3つに分類され（端末入出力閉塞，ディスクあるいは磁気テープ入出力閉塞，ページ入出力閉塞)，実行可状態のプロセスは3つに分類されていることに注意してほしい（CPU 限界， 高順位入出力限界，中順位入出力限界)．スケジューリングの方針は，高順位実行可リストからプロセ

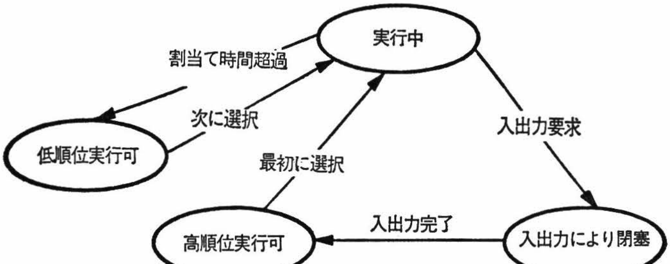


図 4.22 スケジューリングの状態と遷移

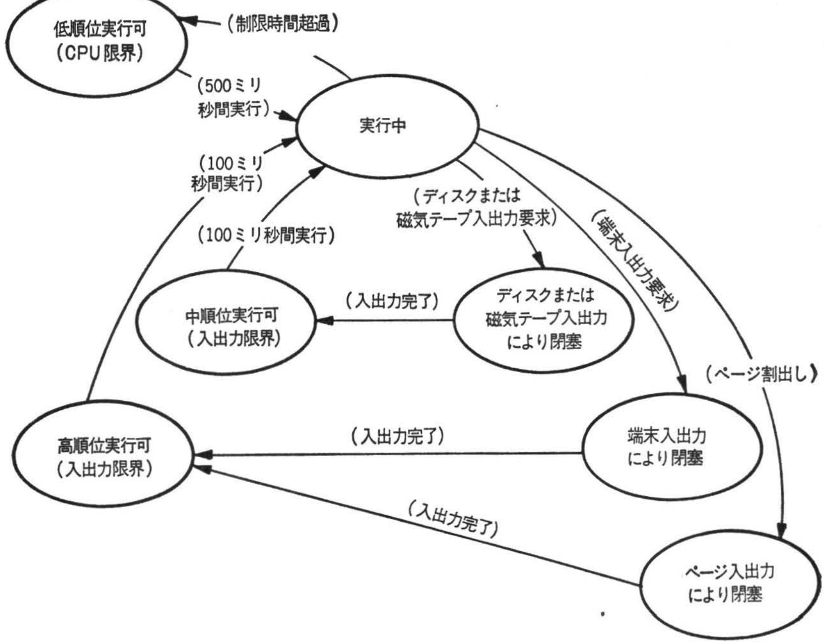


図 4.23 さらに詳細なスケジューリングの状態遷移

スを選択する．高順位のものがなければ，中順位実行可リストからプロセスを選択する．低順位プロセスは，これより順位の高いプロセスが存在しないときに限って実行される．

このアルゴリズムには多くの方針が組込まれている．たとえば，プロセスが入出力要求を出さずに100ミリ秒（約100,000命令）以上実行すると，このプロセスはCPU限界と見なされる．入出力チャネルと入出力装置を動作状態としておくために，入出力限界のプロセスはつねにCPU限界のプロセスよりも先に実行される．すべての入出力限界のプロセスが閉塞されたときには，CPU限界のプロセスに処理装置が与えられ，入出力要求で閉塞されることがなければ，5倍の時間（500ミリ秒）実行を許される．端末入出力が，完了するとプロセスは高順位実行可状態とされることに注意してほしい．このようにするこ

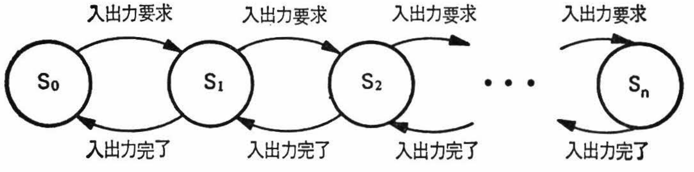
とにより，会話が終了すると直ちに“会話型”ジョブの優先実行を行なうことができる．

別のスケジューラの方針は第9章の議論の一部として示されている．

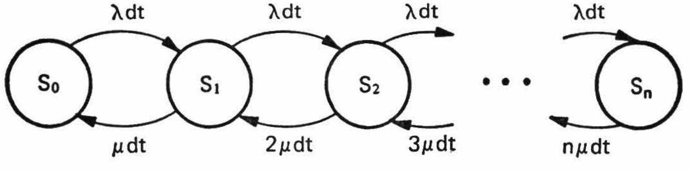
4.3.4　ラウンド・ロビン方式の多重プログラミングの性能評価

状態遷移図は，図 4.24 a に示したように，システム全体の状態を示すモデルとして使用することができる．システムにn個のプロセスが存在するとする．状態 S_i は i 個のプロセスが入出力待合せ（入出力で閉塞）を行なっている状態とする．状態 S_0 では，閉塞状態のプロセスは存在せず，$n-1$ 個のプロセスが実行可で1個のプロセスが実行中である．一方，状態 S_n では，n個のプロセスがすべて入出力の完了を待合せており，実行可のプロセスが存在しないので処理装置はアイドルとなっている．

状態が S_n となる確率 P_n を求めることができるであろうか．できる．図 4.24 a の状態遷移図は 図 4.24 b に示したような**生死のマルコフ過程**に変換することができる（Drake, 1967）．図の遷移の弧に次の時刻での発生確率を記入する必要がある．λdt を次の時刻に実行中のプロセスが入出力要求を出す確率とする（$1/\lambda$ は平均の入出力の要求間隔と解釈できる）．同様に μdt を次の時刻に入出力が完了する確率とする（$1/\mu$ は入出力要求が完了する平均時間と解釈できる）．



(a) 状態遷移



(b) 生死のマルコフ過程のモデル

図 4.24　システム状態のモデル

S_2 から，S_1 への弧には μdt でなく $2\mu dt$ と記されていることに注意してほしい．この理由は，2つの入出力要求は別々のチャネルで実行されていると仮定しており，それぞれの要求は μdt の確率で完了するからである．したがって，次の時刻に**いずれか一方**の要求が完了する確率は $\mu dt+\mu dt=2\mu dt$ である．同様の理由によって，S_i から S_{i-1} への弧には $i\mu dt$ が記入されている．

λ と μ が前の単一プログラミングの場合の入出力待合せ率 ω とどのような関係になっているであろうか．次の関係が容易に求められる：

$$\omega=\frac{\lambda}{\mu+\lambda}$$

入出力要求率 λ が大きいと，入出力待合せ率が大きくなる．入出力装置が高速であれば μ が大きく，入出力待合せ率が小さくなる．

数学的には簡略化しすぎる危険があるが，次の2つの事実を用いて，生死のマルコフ過程を解き，各状態 p_i になる確率 P_i を求めることができる：

1. $S_i \to S_{i+1}$ の遷移（$p_{i,i+1}$）の回数は，$S_{i+2} \to S_{i+1}$ の遷移（$p_{i+1,i}$）の回数に等しい（すなわち，$p_{i,i+1}P_i=p_{i+1,i}P_{i+1}$）．図 4.24 に対してはこのことから次の関係が成立する：

$$\lambda dtP_0=\mu dtP_1$$
$$\lambda dtP_1=2\mu dtP_2$$
$$\lambda dtP_2=3\mu dtP_3$$
$$\cdots\cdots$$

2. P_i の総和は1に等しい．すなわち，

$$\sum_{i=0}^{n} P_i=1$$

これらの2つの事実から，すべての n 個のプロセスが入出力の完了を待合せていて，処理装置がアイドルになっている確率 P_n を求めることができる．まず，次の関係が求められる：

$$P_1=\frac{\lambda dt}{\mu dt}P_0=\left(\frac{\lambda}{\mu}\right)P_0$$

$$P_2=\frac{\lambda dt}{2\mu dt}P_1=\left(\frac{\lambda}{2\mu}\right)P_1=\left(\frac{\lambda}{2\mu}\right)\left[\left(\frac{\lambda}{\mu}\right)P_0\right]=\frac{1}{2}\left(\frac{\lambda}{\mu}\right)^2P_0$$

$$P_3=\frac{\lambda dt}{3\mu dt}P_2=\left(\frac{\lambda}{3\mu}\right)P_2=\left(\frac{\lambda}{3\mu}\right)\left[\left(\frac{\lambda}{\mu}\right)^2\frac{P_0}{2}\right]=\frac{1}{3\times 2}\left(\frac{\lambda}{\mu}\right)^3P_0$$

一般に

$$P_i=\frac{1}{i!}\left(\frac{\lambda}{\mu}\right)^i P_0 \qquad [4\text{-}1]$$

$$\sum_{i=0}^{n} P_i=1$$

が成立する場合には

$$P_0=1-\sum_{i=1}^{n} P_i$$

式 [4-1] から次式を得る：

$$P_0=1-\sum_{i=1}^{n}\frac{1}{i!}\left(\frac{\lambda}{\mu}\right)^i P_0$$

上式を変形すると

$$P_0=1-P_0\left[\sum_{i=1}^{n}\frac{1}{i!}\left(\frac{\lambda}{\mu}\right)^i\right]$$

$$P_0\left[1+\sum_{i=1}^{n}\frac{1}{i!}\left(\frac{\lambda}{\mu}\right)\right]=1$$

$$P_0=\frac{1}{1+\sum_{i=1}^{n}\frac{1}{i!}\left(\frac{\lambda}{\mu}\right)^i} \qquad [4\text{-}2]$$

$$\frac{1}{0!}\left(\frac{\lambda}{\mu}\right)^0=1$$

であるから

$$1+\sum_{i=1}^{n}\frac{1}{i!}\left(\frac{\lambda}{\mu}\right)^i$$

は次のように書ける：

$$\frac{1}{i!}\sum_{i=0}^{n}\left(\frac{\lambda}{\mu}\right)^i$$

したがって，式 [4-2] は次のようになる：

$$P_0=\frac{1}{\sum_{i=0}^{n}\frac{1}{i!}\left(\frac{\lambda}{\mu}\right)^i} \qquad [4\text{-}3]$$

式 [4-1] を用いると

$$P_n=\left(\frac{\lambda}{\mu}\right)^n\frac{P_0}{n!}$$

P_0 の値として式 [4-3] を用いると次式を得る：

$$P_n=\frac{\left(\frac{\lambda}{\mu}\right)^n}{n!\sum_{i=0}^{n}\frac{1}{i!}\left(\frac{\lambda}{\mu}\right)^i} \qquad [4\text{-}4]$$

$$\omega=\frac{\lambda}{\mu+\lambda}$$

であることから

$$\left(\frac{\lambda}{\mu}\right)=\left(\frac{\omega}{1-\omega}\right)$$

が成立し，次式を得る：

$$P_n=\frac{\left(\frac{\omega}{1-\omega}\right)^n}{n!\sum_{i=0}^{n}\frac{1}{i!}\left(\frac{\omega}{1-\omega}\right)^i} \qquad [4\text{-}5]$$

P_n は第 3.2.2 で求められたもので図 4.14 に表にして示してある．

読者はこの計算の条件を思い出してほしい．すべてのプロセスは同じ入出力待合せ率 ω を持ち，ラウンド・ロビンのスケジューリングのように全く対等に処理されると仮定していた．もっと複雑なモデルを用いて性能を求めることができるが，数学的処理が複雑となる．(Arden, 1969；Baskett, 1971；Buten, 1972；Coffman, 1968；Hellerman, 1970；Kleinrock, 1970；Mckinney, 1969 を参照)．

4.4 多重処理装置システム

仕事量，信頼性，計算能力，並列処理，および経済性を増すために，システムによっては処理装置が増設されることがある．初期の多重処理システムではこの増設の処理装置は入出力チャネルのように特殊機能を持っていた．後になって，多重処理システムが発展して，1台の大きな CPU と何台かの周辺処理装置を含むものが構成された (CDC 6600)．これらの処理装置は，ディスプレイを動作させるようなきわめて複雑な仕事を果たすことができる．もっと一般

的な多重処理の例は，たとえば，HIS 6180，IBM 158 MP および 168 MP，UNIVAC 1108，Burroughs 6700 のように，同じ能力の処理装置を2台以上用いるものである．多数の異種の計算機が，しばしば遠距離から接続される計算網も存在する．本節では，同一能力の CPU を複数台用いる多重処理装置システムを論じよう．

多重処理システムを接続し，動作させるには次のように種々の方法がある：

1. 別個のシステム
2. 対等のジョブ・スケジューリング
3. マスタ／スレーブのスケジューリング
4. 斉次のスケジューリング

これらの技法はスケジューリングの複雑さが異なっている．次にこれらを順に論じる．

4.4.1 別個のシステム

IBM 360/67 のようなシステムは，論理的にそれぞれ1台の処理装置と主記憶装置および周辺装置を持つ2つ以上のシステムに分割することができる．これは同一の室内に2台以上の別個の計算機システムを備えているのと全く同じことである．

この構成の利点は，処理装置，記憶装置，および入出力装置が，手動ではあるが，容易に切換えられることである．たとえば，それぞれ1台の処理装置と512Kバイトの記憶装置と8台ずつのディスク駆動装置を持つ2台のシステムとすることもできるし，1台の処理装置と1024Kバイトの記憶装置と16台のディスク駆動装置を持つ1台のシステムとすることもでき（この場合，2台目の処理装置は使用されない）そのほかの組合せ方も可能である．

このように構成の融通性があることは，全部の記憶装置とか全部の入出力装置の資源を必要とするジョブが存在するとき有用である．このほかに，1台の処理装置が修理中である場合には，そのほかの資源はいたずらにアイドルとならず，1台の大規模なシステムの中で共同使用することができる．

このような個別のシステムの場合には，処理装置間でのジョブやプロセスのスケジューリングは，手動で行なわれる分を除いては，存在しない．

4.4.2 対等のスケジューリング

個別の多重処理装置の技法の変形が**対等のジョブ・スケジューリング**の方法

である（この方式は**疎に結合した多重処理**と呼ばれる）．各処理装置には別個のシステムが対応づけられる．ジョブが到着したときには，このジョブはどのシステムに割当ててもよい．ジョブを1つのシステムに割当てることは，最も負荷の軽いシステムに割当てるといった，ジョブの資源要求とシステムの方針に基づいてなされる．

このようなバランスを保つためには，すべてのジョブ・スケジューリングを調節しなければならない．これは，特殊目的の計算機では手動で行なわれることもあるし（Lawrence Radiation Labolatory のオクトパス・システムで行なわれている），実際の処理システムで行なわれることもある（IBM OS/VS2 ジョブ入力システムで行なわれている）．

4.4.3 マスタ／スレーブのスケジューリング

対等のジョブ・スケジューリング技法のように，1つのシステムに恒久的にジョブを割当てる方式では，短期間の資源要求の平衡をとることができない．一般的な多重処理装置システムでは，すべての記憶装置やすべての入出力装置はすべての処理装置から使用可能である．記憶装置や入出力装置はプロセスに割当てられたものであり，処理装置に割当てられるものではない．（この方式は**密に結合した多重処理**と呼ばれる）．処理装置は，使用できる処理装置が存在していないとき以外には，プロセス・スケジューラによって指定されたように，プロセスを実行するために割当てられる．

マスタ／スレーブのスケジューリング技法では，1台の処理装置がシステム中のすべてのプロセスの状態を管理し，スレーブの処理装置の関係と同様である．たとえば，マスタの処理装置は実行すべきプロセスを選択し，空いている処理装置を見つけ，処理装置起動命令を実行する．スレーブの処理装置は，指定された記憶番地から実行を開始する．スレーブの処理装置が入出力要求のような例外的事象に出会ったときにはマスタの処理装置に割込みをかけ，次の命令が出されるのを待って停止する．プロセスには前とは別のスレーブの処理装置が割当てられることもあるのに注意してほしい．

図 4.25 は，多重処理装置が多重プログラミングのために使用されている様子を示したものである．通常，幾つかのプロセスが主記憶に入っている．実際には，プロセスの番地空間はページに分けられていることが多く，ある時点では一部だけが主記憶に入っている．このようにすれば多数のプロセスをシステム内でアクティヴにすることが可能となる．

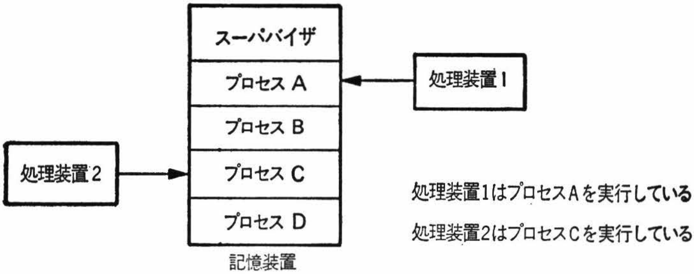


図 4.25 複数の処理装置による多重プログラミング

4.4.4 斉次的な処理装置のスケジューリング

マスタ／スレーブ方式のスケジューリングは，考え方が非民主的であるだけではなく，性能と動作の点で欠陥がある．スケジューリングの仕事が多いときには，マスタの処理装置が過負荷となり重大なあい路となる．もっと民主的な方法はすべての処理装置を対等に取扱うもので，各処理装置はマスタになったりスレーブになったりできる．

このような条件では，処理装置のスケジューリングは論理的には分散される．プロセスとプロセスの状態の一覧表は，共通領域などを使用して格納され，任意の処理装置からこの表を参照することができる．プロセスが入出力待合せやタイマ時間切れなどによって停止したときには，このプロセスを実行していた処理装置はプロセスの状態表でプロセスの状態を更新し，実行すべき別のプロセスを見つける．各処理装置は次に実行すべきプロセスを見つけるに当たって同一のスケジューリング・アルゴリズムを使用する．すべての処理装置が全く同じように取扱われている限り，単一処理装置システムでも，多重処理装置システムでもスケジューリングの方針には差違がない．

分散式のオペレーティング・システムにおいては，処理装置間を十分調整し，円滑に通信を行なう必要がある．第4.5.4節で示すソフトウェア・ロックアウトの例はこの点を表わしている．

4.5 プロセスの同期

プロセス同期の問題は，計算機システム内の資源を共用する必要性から生じる．資源を共用するには，正しい動作を保証するために調整と協調が必要となる．

1つの例で問題点の幾つかを説明しよう．図 4.26 は即席ハンバーガのスタンドである．読者は受付けに注文を出す．受付けは注文書を回転テーブルに置く．店員は注文書を回転テーブルから1つずつ取り，注文の品を袋に入れる．次にそれを会計に渡す．会計は袋を1つずつ包み，代金を受取って読者に注文の品を渡す．このほかに，後ろの部屋には料理人がいて，店員のためにハンバーガをたえず補給している．この例では，各プロセス（人間）はほとんど独立に動作している．通信は，**注文の回転テーブル**，**料理ずみのハンバーガの置場**，および**カウンタ**の**共通領域**だけを通じて行なわれる．

この簡単な例では，当たり前と考えている通信と協同作業の形態を説明している．店員は処理すべき注文が存在するときだけ袋に詰める．ひまで注文がないときには店員はどうであろうか．いねむりをすることにする．客が来て注文が出されたとき店員はどのようにして起こされるか．受付けが店員をけとばす．ここで別の形の通信と協同作業を考えるのは有用な演習である．たとえば，長らくひまだった間に，料理人がスタンドをハンバーガでうめつくしてしまわないようにするにはどのようにしたらよいか．

複数のプロセスが存在する計算機システムでは類似の状況が起こる．場合によっては，資源の不足のためにオペレーティング・システムによって強制的に調整が行なわれることがある．たとえば，入出力チャネルの使用を待合せなければならないことがある．別の場合には，1つのジョブが共に動作する幾つかのプロセスから構成されていることもある．航空機の座席予約システムのような情報検索システムでは，ハンバーガ・スタンドの例のように，別々の仕事を行なう幾つかのプロセスが存在する．1つの仕事は要求を受付け，入力のデー

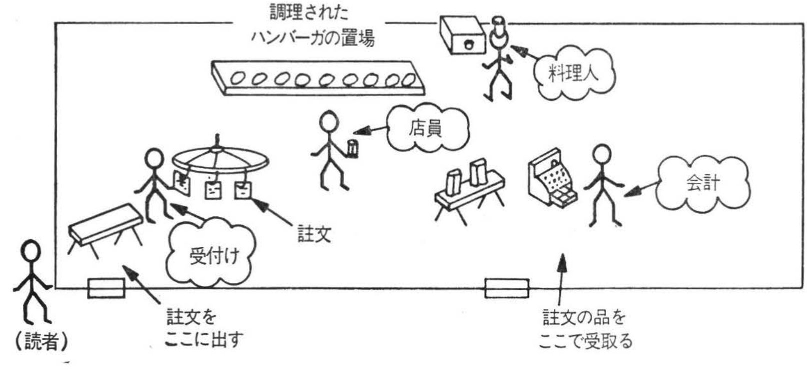


図 4.26 ハンバーガ・スタンドの動き

タを検索し，この要求を“作業者”のプロセスに渡す．作業者プロセスは必要な情報を探すためにデータにアクセスし，それを“出力”プロセスに送る．出力プロセスはこの回答を元の要求者に返す．

処理装置の割付けとプロセス間通信に関しては，競争条件とすくみ（dead lock）の2つの同期の問題がある．

4.5.1 競争条件

競争条件は2つのプロセスのスケジューリングが非常に微妙で，スケジューリングの順序によって計算結果が異なる場合に発生する．競争条件は，2つ以上のプロセスでデータや資源を共用するとき（陽にまたは陰に）に起こる．図4.27は単純な競争の型を示している．

図4.27では，2つのプロセスは実行中である（多重プログラミングされている）．各プロセスは時々1台の印刷機に行を印刷することを要求する．プロセス1と2のスケジューリングによって，プロセス1の印刷結果がプロセス2よりも先になったり後になったりする．しかしもっと起こりそうなことは各プロセスの印刷出力が紙の上で混ざってしまうことである．

このような状態を解決する1つの方法は，共用資源（この場合印刷機）を使用する前にプロセスが陽に**要求**を出すことである．この資源の使用が完了したときには（たとえば，すべての印刷が完了したとき），この資源はプロセスによって**解放**される．**要求**と**解放**の動作は，通常，トラフィック制御で処理されるオペレーティング・システムの機能の一部である．プロセスがすでに使用中の資源を要求したときには，このプロセスは自動的に閉塞状態にされる．後刻，この資源が解放されて使用可能となったときには，閉塞されたプロセスにこの資源が割当てられて実行可とされる．印刷機に対する別の解決法は（たとえば，スプーリング）第5章で論じられる．

物理的な装置のほかに，共通の表（図4.26の註文書の回転テーブルに類似する）のような別の共用資源が存在して，同様のプロセス同期を必要とする．

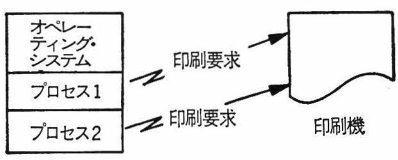
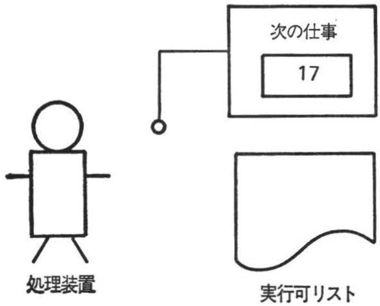


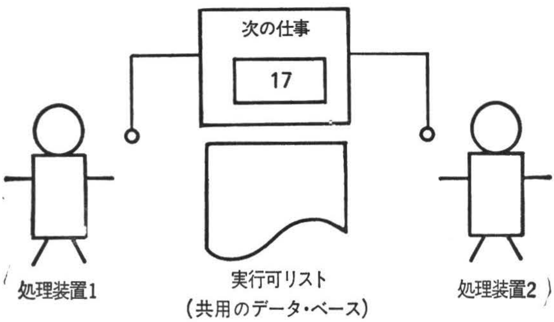
図4.27 単純な競争条件

このような競争条件の例として，図 4.28 を考えてみよう．図は実行可リストからプロセスを選択し実行状態とする手続きを示している．このアルゴリズムは単一処理装置の場合にはうまくいくが，多重処理装置の場合には 図 4.29



現在のプロセスが完了したり実行できなくなったとき：
1 実行可リストの中で現在のプロセスを探す
2 これに”完了” または”延期”と印をつける
3 ”次の仕事”は17番であることを知る
4 プロセス17を実行可リストの中で探す
5 このプロセスを実行すると記入する
6 実行可リストを調整して”次の仕事”を18番とする

図 4.28 スケジューリング（単一処理装置）



2つの処理装置が同時にプロセスを完了したとする：
1 処理装置1は実行可リストの中で実行していたプロセスを探す
2 処理装置2は実行可リストの中で実行していたプロセスを探す
3 処理装置1はそのプロセスに印をつける
4 処理装置2はそのプロセスに印をつける
5 処理装置1は”次の仕事”は17番であることを知る
6 処理装置2は”次の仕事”は17番であることを知る
7 処理装置1はプロセス17を実行可リストの中で求めて印をつける
8 処理装置2はプロセス17を実行可リストの中で求めて印をつける
9 処理装置1は実行可リストを調整して”次の仕事”を18番とする
10 処理装置2は実行可リストを調整して”次の仕事”を19番とする
いまや，両方の処理装置は同時に同じプロセスを実行し，プロセス18は飛ばされてしまった

図 4.29 スケジューリング（複数の処理装置）

に示したように混乱に陥る．

別々の処理装置で実行されている2つのプロセスがほとんど同時に閉塞されたとしよう．そのプロセスの状態を保存した後で，各処理装置は次に実行するプロセスを決定する．両方の処理装置は同じアルゴリズムを互に独立に用いているので，同じプロセスを選択するであろう．さらに，選択の条件を調節するときに，図 4.29 に示したように1つのプロセスを飛び越してしまい，実行可リストを目茶苦茶にしてしまう可能性がある．

この問題点は，共通のデータベースを非同期で，複数の処理装置によって処理するときに起こることである．通常オペレーティング・システムには，記憶割付けマップやページ割付けマップ，入出力表などの共通のデータベースが多数存在している．競争条件とその防止の例は第2.5.2節の入出力がバッファリングの問題で示されている．その他の例は第7章の例題のオペレーティング・システムにおいて見ることができる．

競争の問題を解決するためには，通常独立に動作している処理装置は共通のデータベースを参照するとき同期をとる必要がある．図 4.30 はこの機構を説明している．

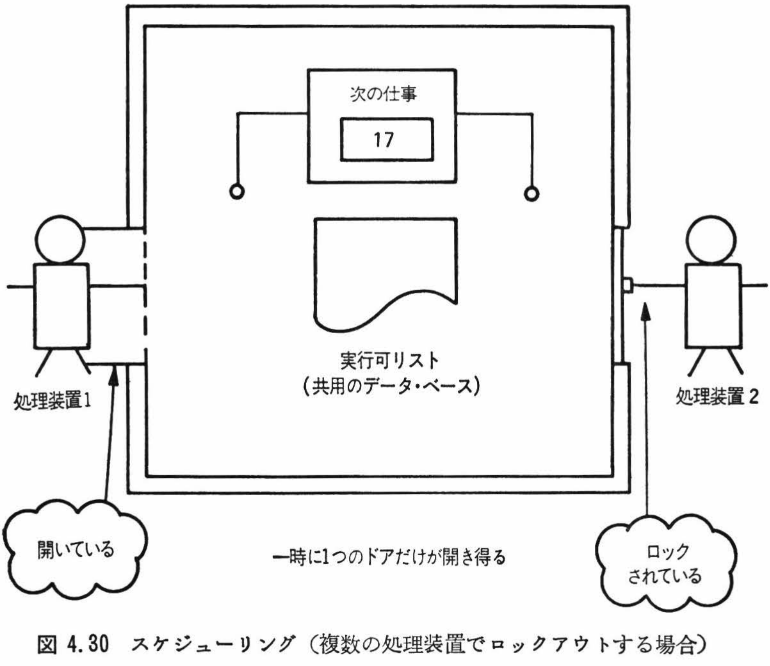


図 4.30　スケジューリング（複数の処理装置でロックアウトする場合）

実行可リストのデータベースを参照する前に，処理装置は特別の“ロック・ビット”を検査する．このビットが設定されていなければ，データベースは使用中でない．この場合には，処理装置はビットを設定してデータベースの処理を開始することができる．処理装置が処理を終了したときには，このビットをオフにする．ロック中に第2の処理装置がデータベースの参照を要求すると，この処理装置はロック・ビットが設定されていることを見出し，ロックが解かれるまで待合せる必要がある．この状態のとき，第2の処理装置は**ソフトウェア・プロセッサ・ロックアウト**となっているといい，一時的にアイドルとなる．第2，第3，第4と任意の台数の処理装置は同時にこのような状態となり得る．

このような場合には協調が必要であることに注意してほしい．もしプロセスが割付けられた資源を解放するのを“忘れた”ならば，この資源は再び他のプロセスによって使用することができないであろう．このために，通常オペレーティング・システムは，プロセスが終了したとき，割付けていた資源をすべて解放する．

4.5.2 同期の機構

プロセス間の協同作業と通信を行なうためには種々の同期の機構を使用することができる．本節では最も一般的な技法を幾つか簡単に述べる．詳細および別法については文献を参照してほしい (Dijkstra, 1965, 1968 ; Easton, 1971 ; Haberman, 1971 ; Hansen, 1970 ; Liskov, 1972 ; Saltzer, 1966).

4.5.2.1 検査設定命令

たいていの同期の方式では，資源の状態を表わすのに物理的な情報が使用される．これは**ロック・バイト**とか**セマフォア**とか呼ばれる．共用されるデータベースごとに別個のロック・バイトが存在する．本書では，ロック・バイト＝0は資源が使用可能であることを示すことにし，ロック・バイト＝1は資源がすでに使用中であることを示すことにする．このような共通資源の処理を行なう前に，プロセスは次の動作を行なわなければならない：

1. ロック・バイトを検査する（0か1か)．
2. ロック・バイトを1にする．
3. ロック・バイトが1であったならば手順1に戻る．

プロセスが資源の利用を完了したときには，ロック・バイトを0に設定する．

ロック・バイト X を持つ共用資源の利用前に行なう動作を LOCK (X), 利用後に行なう動作を UNLOCK (X) と書こう．これらの動作は REQUEST (X) および RELEASE (X) ともいわれる．これらの動作は，IBM 370 では次の命令によって行なわれる：

```
LOCK (X)            UNLOCK (X)

TS    X             MVI   X,'00'
BNZ   *-4
```

ロックを行なう手順1および2の間で，ロック・バイトが他のプロセスによって変更されることがないということが基本的に重要なことである．この要件は，IBM 370 では，手順1と2を1動作で遂行する検査設定命令（TS）を用いて行なわれる．今日のたいていの計算機では同様の命令が備わっている．

読者はこのロックとアンロックの動作によって競争条件を防止できることを確認してほしい．例として次の3つのプロセスを考えよう：

```
プロセス 1               プロセス 2            プロセス 3
   :                        :                     :
LOCK (X)                 LOCK (X)              LOCK (X)
      (微妙なデータ・
   :   ベースの処理)         : (...)               : (...)

UNLOCK (X)               UNLOCK (X)            UNLOCK (X)
   :                        :                     :
```

ロック・バイトは最初0である．すべての時間関係について考察してみよ．

4.5.2.2　待合せおよび信号の機構

上述の機構は同期の目的には十分であるが処理装置という資源を無駄にしている．閉塞されたプロセスは実際には停止せず，たえずループを回ってロック・バイトを調べ，0に変わるのを待合せている．閉塞されたプロセスに対してこのように資源を浪費するのは好ましくない．このために，ロックおよびアン

ロックの機構を次のように変更する：

LOCK (X)：

1. ロック・バイトを検査する（0か1か).
2. ロック・バイトを1に設定する.
3. ロック・バイトの値が1であったならば WAIT (X) を呼出す.

UNLOCK (X)：

1. ロック・バイトを0に設定する.
2. SIGNAL (X) を呼出す.

WAIT および SIGNAL は処理装置管理のトラフィック制御部分の基本機能である．WAIT (X) はプロセスのプロセス制御ブロック (PCB) を閉塞状態に設定し，ロック・バイト X に PCB を結合する．次にプロセス・スケジューラによって他のプロセスが選択され実行される．SIGNAL (X) はロック・バイト X に関連する閉塞プロセスのリストを調べ，X を待合せているプロセスが存在すれば，1つのプロセスを選択し，その PCB を実行可状態に設定する．最後にプロセス・スケジューラは，この新しく“起こされた”プロセスを選択して実行させる．

WAIT および SIGNAL の機構は，入出力待合せなどの別の目的のためにも使用することができる．入出力要求 (SIO 命令) が発せられた後に，プロセスは WAIT (X) で閉塞させることができる．ここで X は入出力装置に関する状態バイトである（例題のオペレーティング・システムで示した装置制御ブロックの中にロック・バイトを入れることもできる)．入出力完了割込みが発生すると，割込みは SIGNAL (X) に変換される．

4.5.2.3 計数セマフォアに対するPおよびVオペレータ

もっと一般的な LOCK/UNLOCK の機構は，Dijkstra (1968) によって定義されたPおよびVオペレータである．PおよびVは，0および1だけではなく，整数値をとる**計数セマフォア**に対して作用する．この機構は次のように定義できる：

P (S)：

1. Sの値を減じる（S＝S－1).
2. Sが負のとき，WAIT (S).

V (S)：

1. Sの値を増す（S＝S＋1).
2. Sが0または負のとき，SIGNAL (S).

セマフォアの初期値と個数によって，PおよびVは多くの目的に使用することができる．1つの初期値1のセマフォアを用いると，P (S) および V (S) は WAIT (S) および SIGNAL (S) と同じである．

これらの機構の利用としてもっと興味のあることは，“生産者と消費者”の問題である (Dijkstra, 1968；Denning, 1971)．生産者は，n 個の品物を収容できる共用のバッファの中に消費者が使用する物を入れる．このよい例は，第2.5.2節で用いた出力バッファの処理や，図4.26の“ハンバーガの置場”である．生産者は一杯のバッファに品物を入れることはできない．また，消費者は空のバッファから品物を取出すことはできない．初期値 $S_1=n$ および $S_2=0$ の2つのセマフォアを用いることによって，次のようにしてプロセスの同期をとることができる：

```
生産者プロセス, Pp
PROCEDURE:
        :  (製品を生産する)
   P  (S1)
        :  (製品をバッファに入れる)
   V  (S2)
   GO TO PRODUCE
```

```
消費者プロセス, Pc
CONSUME:
   P  (S2)
   :  (製品をバッファ
       から取出す)
   V  (S1)
   :  (製品を消費する)
   GO TO CONSUME
```

読者はここでも正しく同期がとれることを検証してみてほしい．図4.31のように模擬的に実行してみるとセマフォアによる調整の作用がよく理解できる．図では $n=2$ である．生産者は一杯のバッファに品物を入れることができないし，消費者も空のバッファから品物を取出すことができないことに注意してほしい．

複数の生産者と消費者がいる場合にも同じアルゴリズムを使用することができる．セマフォアによる同期の別の例は例題のオペレーティング・システムの中で説明する．

4.5.2.4 メッセージの通信

これまで論じてきた同期機構では，共用のバッファ領域や共用のロック・バ

	動作		セマフォアの値	
事象	P_p(生産者)	P_c(消費者)	S_1	S_2
0	...	...	2	0
1	—	$P(S_2)$ 閉塞	2	-1
2	$P(S_1)$	— (バッファ"空")	1	-1
3	$V(S_2)$ 閉塞を解く P_c	—	1	0
4	$P(S_1)$	—	0	0
5	—	$V(S_1)$	1	0
6	$V(S_2)$	—	1	1
7	$P(S_1)$	—	0	1
8	$V(S_2)$	—	0	2
9	$P(S_1)$ 閉塞	—	-1	2
10	— (バッファ	$P(S_2)$	-1	1
11	— "一杯")	$V(S_1)$ 閉塞を解く P_p	0	1

図 4.31 2つの同期したプロセスの模擬実行

イトあるいはセマフォアによって間接的なプロセス間通信を行なうことができた. 生産者と消費者の問題のような場合には, SEND (P_r, M) および RECEIVE (P_s, M) という基本機能を用いることによって直接的なプロセス間の通信を行なうことができる. ここで, P_r および P_s はプロセスの名前であり, M はメッセージ（k バイトの文字列）である.

SEND (P_r, M) は, 受信者 P_r のためにメッセージ M を保存する. RECEIVE (P_s, M) は, 要求したプロセスに対するメッセージが届いていれば, このメッセージ M および送信者の名前 P_s を返却する. もしメッセージが存在しないときには, メッセージが送られてくるまで閉塞状態となる. メッセージのバッファリングと同期は, SEND および RECEIVE 機能によって自動的に処理される.

例題のオペレーティング・システムでは, メッセージ通信の詳細な組込みとその利用例を多数示している.

4.5.3 すくみ

すくみとは, 2つのプロセスが互に他が確保しているので自分が使用することのできない資源を, それと知らずに待合せるような状態をいう. 図 4.32 に示された2つのプロセスを考えてみよう.

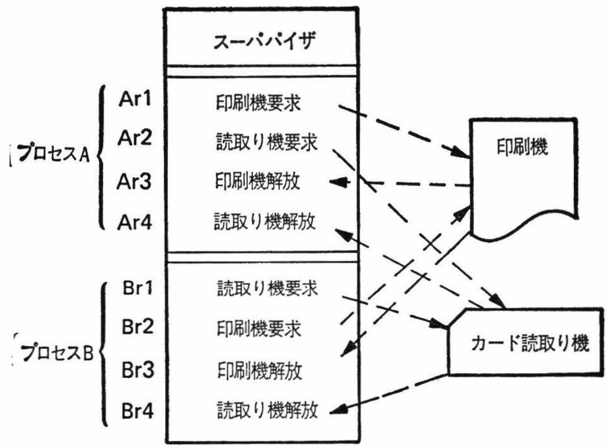


図 4.32 すくみの状況

プロセスAおよびBは，要求と解放の動作によって印刷機とカード読取り機とを共用している．プロセスは独立にスケジュールされているから，要求と解放の動作は次に示したように種々の順序で混ざることがある：

(1) Ar1, Ar2, Ar3, Ar4, Br1, Br2, Br3, Br4
(2) Br1, Br2, Br3, Br4, Ar1, Ar2, Ar3, Ar4
(3) Ar1, Ar2, Br1, Ar3, Ar4, Br2, Br3, Br4

読者はこれらの順番が実際に起こり得るものであり，正しく動作することを確かめてみてほしい．第3の順序で，プロセスBはプロセスAがAr4を発するまでBr1によって一時的に閉塞されることに注意してほしい．

Ar1（プロセスAが印刷機を要求）およびBr1（プロセスBが読取り機を要求）で始まる順番を考えてみよう．次に何が起こるであろうか．もしAr2（プロセスAが読取り機を要求）が起これば，読取り機はすでにプロセスBで使用中であるから，プロセスAは閉塞状態とならなければならない．次にBr2（プロセスBが印刷機を要求）が起こると，印刷機はすでにプロセスAで使用中であるから，プロセスBも閉塞状態とならなければならない．このようにして各プロセスが，他のプロセスが使用している資源を必要として，それぞれが解放されるのを待合せるためにすくみの状態となることがある．

多重プログラミングのシステムにおいては，すくみの問題は種々の場合，特に表を共用するときにはよく起こる可能性がある．

すくみを処理するために次の技法を論じよう：

1. 共用資源をあらかじめすべて割付ける．
2. 制限を設けて割付ける．
   a．被制御割付け
   b．標準型の割付け
3. すくみを検出して回復する．

4.5.3.1 共用資源をあらかじめすべて割付ける

すくみを防止するための安全で簡単な方法は，ジョブを提出するときに，使用しようとする装置や資源を利用者にすべて宣言させることである(Coffman, 1970；Fontao, 1971)．スケジューラは単純に，ジョブが必要とする資源のすべてが空いていないとジョブをスケジュールしない．資源がジョブに割当てられると，実行の間中保有される．この技法は IBM OS/360 において装置割付けで使用されているが，幾つかの欠点がある：

1. ジョブが使用する装置の全部が実行する前にわからないことがある．
2. 資源の一部は，一時間も後の実行の終了時まで必要でないかもしれないのにすべての資源が空くのを待合せなければならない．
3. ジョブでその資源を使用する場合がほとんどないときでも装置をジョブに割付けておくのはシステムにとって無駄である．たとえば，ジョブの主記憶イメージ印刷機にダンプするのは，誤りが起こったときだけに必要となる．しかも，この技法では，ジョブに印刷機を割付ける必要がある．
4. 1つのジョブにとっては，そのジョブ実行の全期間を通してすべての装置が必要となるわけではない．多重プログラミングの場合には，他のジョブが使用できるように資源を解放することも可能である．

4.5.3.2 すくみを防止するための制限つき割付け

被制御割付け (Haberman, 1969；Dijkstra, 1968)

第4.5.3.1節と同様に利用者が必要とするすべての資源をあらかじめ指定する．ただし，すべての資源が空いていなくてもスケジューラはジョブを開始させる．この方法の考え方は，すくみの起こる可能性のある場合にはジョブに資源を割付けないようにするものである．たとえば，システムには9台の磁気テープ装置があり，ジョブ1とジョブ2が最大6台の磁気テープを必要とする場

合を考えてみよう．これらのジョブでは，いつも6台すべての磁気テープを必要とするわけではない．IBM OS/360ではこれらのジョブを一諸には実行しない．しかし被制御割付けを行なうと，図4.33に示したような状態になるまでは，両方のジョブを実行することができる．両方のジョブが2本線を越えて進行すると，すくみが起こる可能性が生じる．この方法では，磁気テープが解放されるまでは，1つのジョブだけが2本線を越えて進行することができる．

もし，ジョブ1に4台の磁気テープを，ジョブ2に5台の磁気テープを割当てたならば，両方のジョブは共に終了することができないであろう．このようなことを防止しなければならない．ジョブ2が3本目の磁気テープを要求する前にジョブ1が完了するか，あるいはジョブ1が最初の磁気テープを要求する前にジョブ2が磁気テープを3本解放する点まで達していれば，どちらのジョブも全く待合せる必要がないことに注意してほしい．したがって，この方法では，ジョブは次に述べることを実行する必要がある：

1. あらかじめ必要最大数の資源を宣言する．

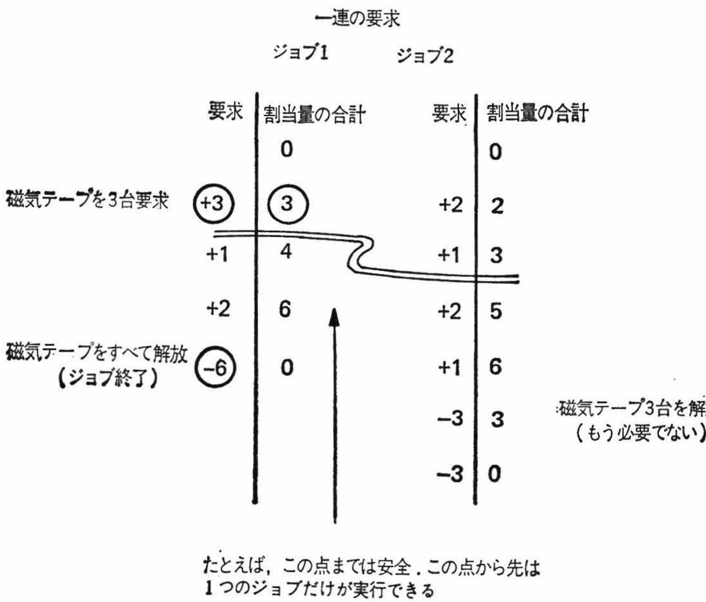


図4.33　被制御割付けの例

2. 資源が割当てられる前に，すくみが起こる可能性を判定する手順を実行する．その可能性がなければ資源を割当てる．図 4.34 はこの手順を遂行するアルゴリズムを示している（Haberman, 1969).

被制御割付け法の問題は，依然として，あらかじめ必要となる最大数を知る必要があることである．そのうえに，このアルゴリズムはあまりにも慎重である．実際には，ジョブは最大量を必要としないことがあり，ジョブの待合せは不要であるかもしれない（2つのジョブが2本線を越えて進行しても，すくみは起こらなかったかもしれない).

標準型の割付け（Havender, 1968)

制限つき割付けの別の方法は標準型の割付けを行なうものである（Havender, 1968)．この方法では，すべての資源に一意的な番号が割当てられている（たとえば，読取り機=1，印刷機=2，穿孔機=3，磁気テープ=4，ディスク=5，…)．割付け要求はすべて**番号が大きくなる順番**でなければならない．

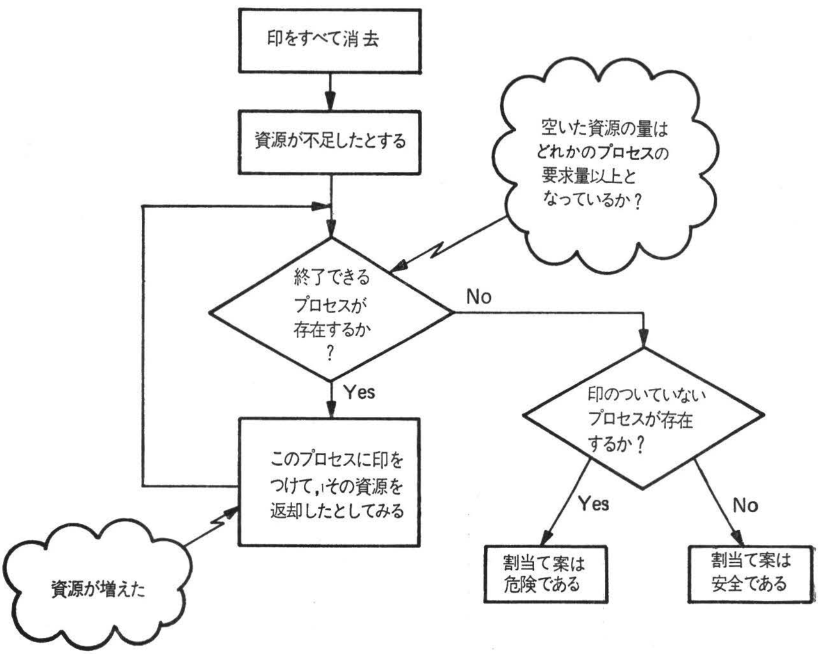


図 4.34 被制御割付けのアルゴリズム

たとえば，読取り機（1），穿孔機（3），磁気テープ（4)の順は正しい要求の順序であるが，読取り機（1），磁気テープ（4），穿孔機（3）の順は正しくない．番号が大きくなっている限り，資源が空いていればこの割付け要求は認められる．このほかには制限事項はない．資源が空いていなければ待合せる必要がある．

この方式ではすくみが発生し得ないことが保証できる．あらかじめ必要量の最大値を指定しないことに注意してほしい．

この手法には幾つかの問題がある．標準型の順番と実際のプロセスの資源要求の順番とは対応しないことがある．磁気テープが今すぐ必要で，印刷機が後で必要となる場合でも，印刷機は磁気テープの要求に先立って要求しておかなければならない．

資源の番号を注意深く割当てると，この標準的な順序はあまり不都合な制限とならないようにすることができる．

一般に，この技法はきわめて限定されたシステムの内部でのみ使用することができる．たとえば，IBM OS/MVT では，内部のシステム表を割付けるためにこの方法が使用されている．しかし，この機能は一般の利用者には利用できない．

4.5.3.3　検出と回復

前述の技法は控え目で慎重にすることによってすくみが起こらないようにしていた．検出と回復の技法は，（1）検出と（2）その回復が可能である限りすくみが起こることもあり得る点で，前に比べもっと勇敢である．すくみの検出はつねに可能ではあるが，方法が難しいときがある．すくみを検出するには次の方法を用いることができる（Bensoussan, 1968；Murphey, 1968)：

資源割当て表（RATBL）

RATBL

資源番号	割当てられたプロセス番号
1	
2	
3	
•	
•	

プロセス待合せ表(PWTBL)

PWTBL

プロセス	待合せている資源
1	
2	
3	
•	
•	

図 4.35　資源とプロセスの状態表

1. 各資源と各プロセスに一意的な番号を任意に割当てる.
2. 資源を確保したとき，プロセスがソフトウェアのロックをかける.
3. 図 4.35 に示すように，資源とプロセスの状態を把握する表を設定する.
4. 資源が確保されたり解放されたりしたとき図の表 RATBL および PWTBL の欄を作成する.
5. ロックされた資源が要求されたときには，図 4.36 に示したすくみ検出のアルゴリズムを用いる.

検出処理を説明するために次の順番を考えてみよう：

資源	プロセス
1	1
2	3
3	2
4	2
5	1

1 P_1 は1_1をロックする

2 P_2 は1_3をロックする

3 P_3 は1_2をロックする

4 P_2 は1_4をロックする

5 P_1 は1_5をロックする

6 P_1 は1_3を要求する

$J=1 \quad I=3$

$K=2$

Kは資源I′を待合せていない→P_1は待合せる

7 P_2 は1_2を要求する

$J=2 \quad I=2$

$K=3$

I′はなし→P_2は待合せる

8 P_3 は1_5を要求する

$J=3 \quad I=5$

$K=1$

$I'=3$

$K'=2$

$2 \neq J$ であるので$K=2$と設定する

$I'=2$

$K'=3$

$K'=J$ ！ → すくみ　P_3を助ける

PWTBL

プロセス	資源
1	3
2	2
3	5

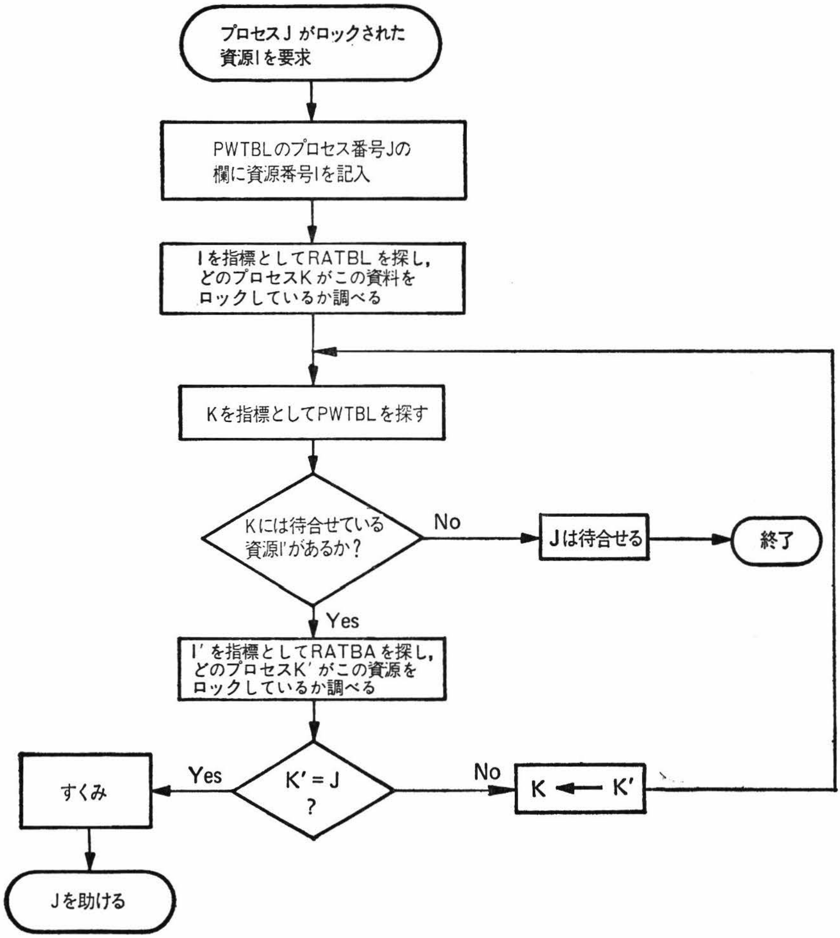


図 4.36　すくみを検出するアルゴリズム

この方法では，資源の要求の順番に何の約束も不要である．資源の解放はトラフィック制御によって監視され，必要に応じ RATBL および PWTBL が適当に変更されて装置が再割当てされる．プロセス J の要求を検査するために，ロックされた資源が要求されるたびに（検出アルゴリズムの入口が呼ばれる）スーパバイザによってすくみの検査が完了するまで，RATBL および

PWTBL をロックする必要がある．その後で次の要求の処理を行なうことができる．もしロックをかけないと，次の要求者 J' が表を変更し，スーパバイザ（まだ J を調べている）がすくみであると判定して，J' ではなく，プロセス J の回復を行なうことになるかもしれない．このように要求を行列させると，スーパバイザが検査する時間のほかにさらにオーバヘッドを追加することとなる．

すくみを検出したときにはそれを解除しなければならない．このことは，原理的には，すくみを解くためにジョブの1つが1つ以上の資源を解放**しなければならない**ことを意味する．回復の問題に対しては一般的な解が存在しない．資源を**安全に**解放するには，ジョブはその資源が割付けられた時点以前に**さかのぼって逆計算**しておかなければならない．元に戻ることは困難なことがある．このための技法は次のものがある：

1. 逆計算アルゴリズム（計算と状態を元に戻す）——ルーチンごとに特別のルーチン．
2. 以前の状態のスナップ（チェックポイント）をとり，割付け前の状態を回復するために使用する．

このチェックポイント技法は，実時間の計算や長時間のデータ処理で誤り回復のために必要とされるので，一般に行なわれている．

検出と回復の方法に関しては幾つかの問題がある．検出はつねに可能であるが，回復はつねには可能であるとは限らないし，経済的に利用可能であるとも限らない．回復が可能であるにしても，通常は困難である．

4.5.3.4　すくみ処理の研究

資源を共用する複雑な多重プログラミング・システムが広く使用されるにつれて，多くのシステムではすくみが大きな問題となっている．すくみは多くの研究対象となっている（Coffman, 1971；Easton, 1971；Fontao, 1971；Haberman, 1969；Havender, 1968；Holt, 1972；Murphy, 1968；Needham, 1969）．

4.5.4　同期に関する考察

同期の機構を採用するとプロセスおよびシステム全般の性能に影響がある．本節では2，3の中心的な問題を論じ解析の例を示す．

4.5.4.1 “中断”と“スピン”のロック

第4.4節で述べたように，多重処理装置システムの動作において，2つのロックの状態を考慮しなければならない．1つのプロセスが入出力待合せなどのような通常のPオペレータによって閉塞されるとき，処理装置は他のプロセスに割当てられる．これは，プロセスが“中断”されて処理装置がほかに割当てられることから中断ロックと呼ばれる．

同様なロック機構は，図 4.30 の多重処理装置システムを同期させるために使用される実行可リストで説明したように，オペレーティング・システム自体の中でも使用される．しかし，この場合は前とは多少相違点がある．実行可リストがロックされていることを知ったとき，プロセスはどうしたらよいだろうか．通常は現在のプロセスを中断して，実行可リストからプロセスを選択して実行させる．しかし，この場合は実行可リスト自体がロックされているので他のプロセスを実行させることができない．したがって，プロセスはロックが解かれるまでたえずロックを判定し続けなければならない（たとえば，検査設定命令を用いる)．論理的には，処理装置は“車輪を回し”続けるので，このロックは**スピン・ロック**と呼ばれる．多重処理装置のオペレーティング・システムでは，ほかでもこのようなスピン・ロックを行なわなければならない場合が多い．

4.5.4.2 ソフトウェア・プロセッサ・ロックアウトの解析

第4.5.4.1節で述べたスピン・ロックの状態は**ソフトウェア・プロセッサ・ロックアウト**と呼ばれている(Madnick, 1968)．システム中のプロセス数が増加するにつれて，ソフトウェア・ロックアウトの回数や期間も増大する．本節では，ソフトウェア・ロックアウトのシステム性能に対する影響を解析する．簡潔さと理解しやすさの点から，第4.3.4節で論じたと同様の単純マルコフ・モデルを使用する（図 4.37)．各処理装置は独立であると考える．プロセスが実行されているとき，システムのデータベースにアクセスする必要が生じるまでの時間は，係数 $1/E$ を持つ逆指数分布函数によって表わされる（アクセスからアクセスの時間の期待値は E である)．同様にして，処理装置は期待値 L の間ロックされた状態で実行する．比の値 $L/(E+L)$ は，おおまかにいえば，単一処理装置の構成で多重プログラミングしているときに（プロセッサ・ロックアウトは起こらない）ロックされた状態となる時間の全実行時間に対する割合と考えられる．

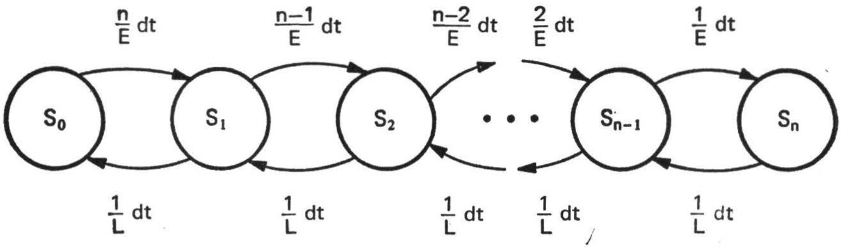


図 4.37 マルコフ・モデル

個々の処理装置は次の3つの状態をとり得る：

1. $1/E$ の割合の時間の間，ロックされない状態でプロセスの実行を行なっている．この時間の後でシステムのデータベースのアクセスを行なう．
2. $1/L$ の割合時間の間，ロックされた状態で実行し，その後ロックが取除かれる．
3. データベースをアクセスしようとしたがロックアウトされてアイドルになる．

図 4.37 で S_i は i 台の処理装置がシステムのデータベースをアクセスしようとしている状態を表わす（1台の処理装置が実際にデータベースを処理し，残りの $i-1$ 台はアクセスを待合せている）．

S_i から S_{i-1} への遷移の確率は $(1/L)dt$ であり，1台の処理装置がロック状態で過ごす時間は，ロックされたデータベースのアクセスを待合せている処理装置の台数には無関係である．S_i から S_{i+1} への遷移の確率は $(n-i)(1/E)dt$ であり，これは実行中の処理装置の内で $n-i$ 台が $(1/E)dt$ の確率でシステムのデータベースのアクセスを要求することを示している．第 4.3.4 節と同様にして，定常状態の確率を求めると次式を得る（Drake, 1967）：

$$P_i = \frac{n!}{(n-i)!}\left(\frac{L}{E}\right)^i P_0 \qquad i=1, \cdots\cdots, n$$

ただし

$$P_0 = \frac{1}{\sum_{i=0}^{n} \frac{n!}{(n-i)!}\left(\frac{L}{E}\right)^i}$$

興味のある性能判定の尺度は，ロックアウトによってアイドルとなる処理装置の台数の期待値である：

$$E(\text{アイドル}) = \sum_{i=2}^{n} (i-1) P_i$$

上式は次のようになる：

$$E(\text{アイドル}) = \frac{\displaystyle\sum_{i=2}^{n} \frac{(i-1)}{\left(\frac{E}{L}\right)^i (n-i)!}}{\displaystyle\sum_{i=0}^{n} \frac{1}{\left(\frac{E}{L}\right)^i (n-i)!}}$$

図 4.38および図 4.39には上式を計算してグラフで示している．たとえば，図 4.39 では，L/E の比が 0.05 のときのソフトウェア・ロックアウトがシステム性能に及ぼす影響を示している．このようなシステムに15台の処理装置が接続されている場合には，ソフトウェア・ロックアウトによって常時 1 台の処理装置がアイドルになる勘定である．処理装置を40台にすると，平均的には19

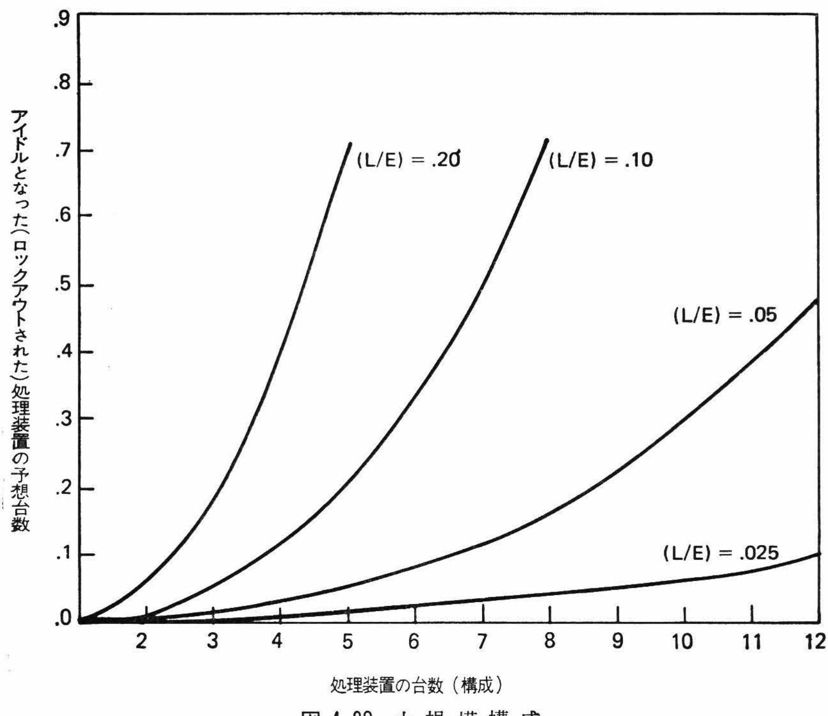


図 4.38 小規模構成

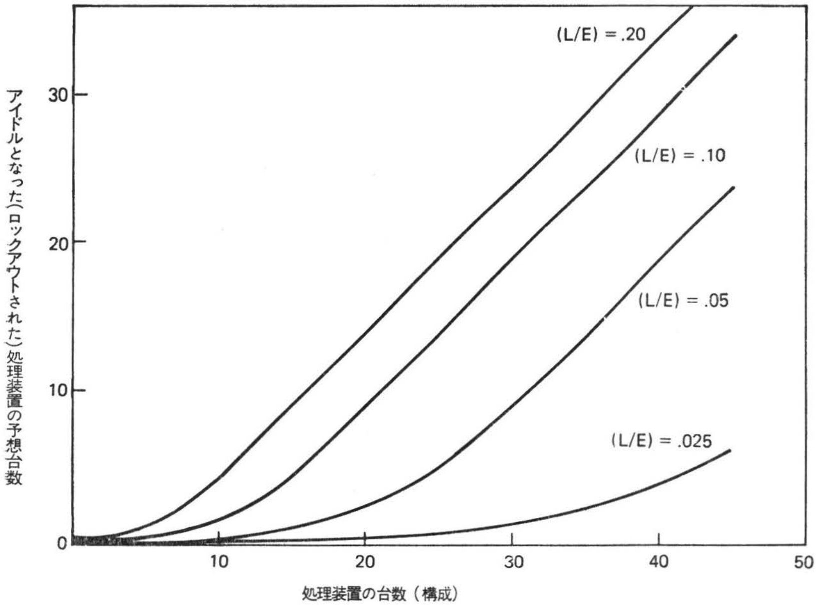


図 4.39 大 規 模 構 成

台がアイドルとなり，41台にすると20台がアイドルとなる．

処理装置が“アイドル”状態であることは注意深く考えてみる必要がある．15台の処理装置は実効的に14台分の処理能力を有し，14台の処理装置は実効的には13.25台分の処理能力を有している．したがって，15台目の処理装置は実効的には75%の能力を有していることになる．このような構成は，経済性が問題のときには追加の処理装置（と関連するハードウェア）の経費が全システムの経費の合計の4%以下の場合，あるいは，追加の計算能力が経済性を無視してでも必要とされる場合には意味のあることである．一方同じ条件の下では，41台目の計算機の能力は0%に近い．

L/E の値はどの程度であろうか．0.1% から 10% の間の値が複雑なシステムで起こりそうである．稼動している単一処理装置のオペレーティング・システムの幾つかでは，実行時間の最大50%をスーパバイザの仕事を行なうのに費している．このようなシステムが多重処理に採用されて，スーパバイザ状態になるとやたらにロックをかけるようになれば，ロック時間 L はロックされない時間 E よりも多くなってしまうであろう．システムは多重処理装置の動作を念頭におき，ロックアウトの発生を最小とし，その時間を短縮するようにして設

計しなければならない．

比 L/E は，実際には，処理装置の台数が増大すると変化することがある．手続き共用の有効性と主記憶容量によって，ページングの負荷は増大することも減少することもある．さらに，システムにおいてより多くのプロセスが作られることになると，ロックされた優先順位のスケジューラを実行するのに必要とされる時間は増大すると考えられる．

本節での簡単なモデルは，多重処理装置のロックアウトの問題全体に対する完全で正確な数学的な解を与えようとするものではない．しかし，量的な解析を行なうことによって，多重処理装置のオペレーティング・システムの設計における1つの因子として，ロックアウトの重要性を理解することができる．

4.5.4.3 同期のオーバヘッドを減少させる技法

スピン・ロックを使用すると，すでに見たように，システムの性能に不利な影響がある．しかし，もっと一般的に使用されている中断ロックを使用すると，ロックを検査し，設定し，あるいは解除するオーバヘッドが増加する．同期のオーバヘッドを減少させるための幾つかの技法があり，それぞれの利点を勘案する必要がある．

1つの手法は単一のロック（またはごく少数のロック）を用いて，共用のデータベースをすべて包含してしまうことである．別の方法は，データベースをすべて注意深く識別して，それぞれにロックを対応させることである．たとえば，実行可リストは2つ以上の実行可リストに分割することもできる（偶数番号のプロセスと奇数番号のプロセス）．このようにしておくとそれぞれの実行可リストに別々のロックを持たせることができる．したがって，2CPU システムでは，1台の処理装置はつねにロックされていない実行可リストを見つけることができる．

精密なロックを用いる方法（多数のロック）と**包括的なロックを用いる方法**（データベースすべてに対して1つのロック）の選択に関しては考慮しなければならない因子が多数存在する．精密法では，特定のデータベースが他のプロセスで使用されることはそんなに多くないとしても，ロックを設定したり解除したりするためのオーバヘッドはかなり大きい．また，ロックが多いとデバッグが大そう複雑となる．

包括的なロック法（腕力法とも呼ばれる）では，長時間（最大50%以上）ロックがオンにされる．このために，ソフトウェア・ロックアウトの確率が増し，処理装置のアイドル時間が大きくなる．

4.6 ジョブとプロセスの混合スケジューリング

多くの場合に，ジョブ・スケジューリングとプロセス・スケジューリングのアルゴリズムは密接に関係し合う必要がある．

あまりにも多くのプロセスを同時に多重プログラミングしようとすると，特にデマンド・ページング・システムでは，システム資源が過負荷となってシステムの性能が悪くなり，すべてのプロセスに被害を及ぼす（この現象は**スラッシング**と呼ばれ，第8章で解析される）．一方，通常のタイムシェアリング・システムでは，利用者はすべてできるだけすみやかに“計算機の割れ目”を要求する．1時間も待つなどは耐えられない．

妥協案として，すべてのジョブにサービスして性能を悪くするよりも，2，3のジョブに多くの機会を与える方がはるかによいという原理を採用することも可能である．たとえば2～5人の利用者のジョブを一時に実行可とすることがある．

本節で用いる基本的な考えは，

1. 2～5ジョブの間の“小数の”ジョブの多重プログラミングで，数ミリ秒ごとにプロセス・スケジューリングする．
2. “多数の”ジョブをタイムシェアリングする．数100ミリ秒から数秒ごとに1つの実行可ジョブが保持状態とされ，保持状態のジョブの内の1つが実行可とされる（図 4.40 参照）．

このようにすると，1分程度の期間で各ジョブに順番が回り，そのうえに，システムもかなり効率よく働く．この技法の変形は，第9章で論じる MULTICS（適格プロセス）や CP/67（Q1 および Q2 の制限）などの，多くのオペレーティング・システムで採用されている．

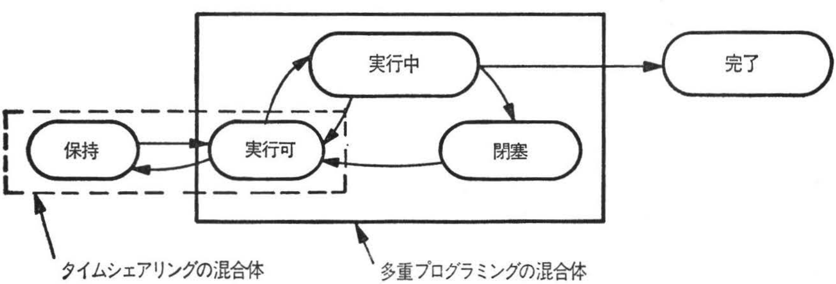


図 4.40 ジョブとプロセスとスケジューリングの相互関係

4.7 要　約

処理装置管理は次に示す主要機能を持っている：

1. ジョブ・スケジューリング
2. プロセス・スケジューリング
3. トラフィック制御（プロセス間通信および同期など）

ジョブ・スケジューラは，手動によるきわめて単純なアルゴリズムから，実行を待合せているジョブの混合体(保持状態)とシステムの空き資源とを考慮する自動的で複雑なスケジューラに至るまで各種存在する．単一プログラミングのオペレーティング・システムや単純な優先順位に基づくプロセス・スケジューラでは，適応性のあるスケジューリング・アルゴリズムで得られるような最適化を行なうことはできない．どのような形のシステムであっても，ジョブ・スケジューリングとプロセス・スケジューリングとの相互関係はつねに注意深く考慮しなければならない．ジョブ・スケジューラが，すでに過負荷となっているプロセス・スケジューラにどんどんジョブを投入すれば，システムの能率は大幅に悪化する．

プロセス間通信と同期に関連した多くの問題を解決するには今後なお精力的な研究が必要である．これらの特性は，第7章の例題のオペレーティング・システムで頻繁に使用して説明しているように，モジュール化された近代的なオペレーティング・システムでは非常に重要なことである．

演　習　問　題

4.1　次の言葉を簡潔に定義し，差違を明らかにせよ．
   1. プロセッサ（処理装置）
   2. プロシージャ（手続き）
   3. プロセス

4.2　プロセスはジョブと等価か，これらが等価であるようなシステムの設計を考えることができるか．等価でないようなシステムはどうか．

4.3　1つのジョブから他のジョブへ切換える（割当て時間によったり別の方法によったりで）ときのオーバヘッドは，すべてのジョブが主記憶に常駐

していたとしても，さほど小さなものではない.

1. プロセスを切換えるとき，プロセスに関するどの情報を変更したり更新したり，保存したりしなければならないか.
2. これらの動作の手助けをするハードウェアが考えられるか.
3. 多重プログラミング・システムにおいて1つのプロセスから他のプロセスへの切換えは，普通，プロセスが割当てられた時間（たとえば，100ミリ秒）を使い果たしたときに起こる．プロセスを切換えるための他の理由あるいは判定規準は何か.

4.4 多重処理システムにおいては，処理装置とプロセスの1対1の対応を必要とせずに，処理装置を任意のプロセスに割当てることができる理由を示せ.

4.5 Honeywell 800 計算機の設計では，各プロセスの状態を，プロセスに関するプログラム・カウンタや状態情報などを一定長の特別なレジスタ表の1つの欄を割当てて格納することにより，効率のよい多重プログラミングを行なおうとしている．切換えは，ラウンド・ロビン方式で表を走査するハードウェア機能を用いて1つのプロセスについて1命令の実行によって遂行される．このような方式を用いることにして次の問に答えよ（ハードウァアの変更をしてもしなくてもよい).

1. ジョブやプロセスをスケジュールするための優先順位方式を組込むにはどのようにしたらよいか.
2. レジスタ，CAW，CSW，その他を退避したり復帰したりする問題はどのように処理したらよいか.
3. 入出力のために閉塞されたプロセスをどのようにして送り出すか.
4. プロセス表の長さが固定で n 欄であるとき，$n+1$ 以上のアクティヴ・プロセスを作ることができるか．それはどうすればよいか.

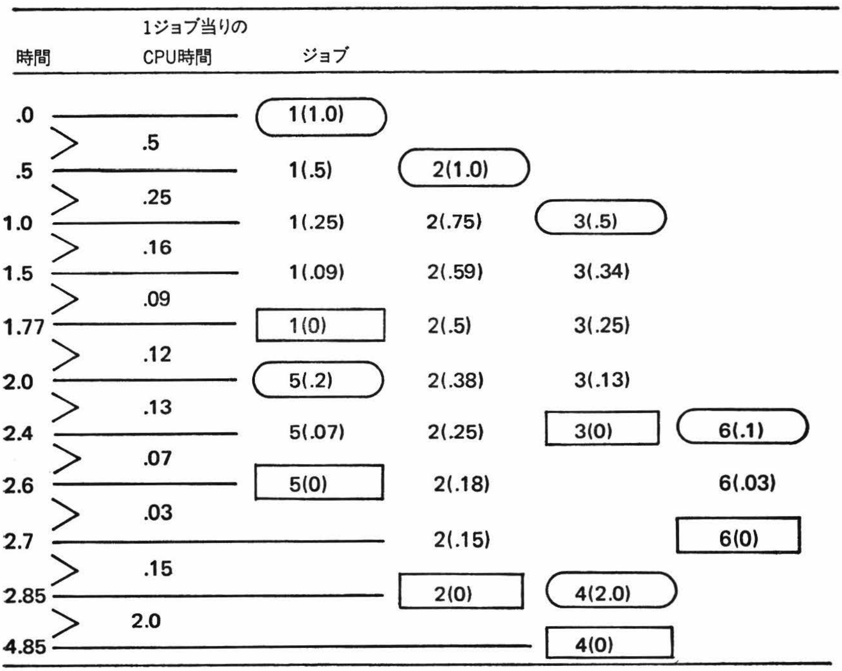
4.6 記憶管理方式を分割割付け方式から再配置型分割方式に変更するときのあつれきを，6つのジョブの処理を調べて，考察してみよう．各ジョブには特定の優先順位がつけられており，記憶域の要求と CPU の要求が行なわれる．簡単のために，すべてのジョブは主として計算を行ない，入出力は少ないと仮定しておく．したがって，2つのジョブが多重プログラミングで実行されたときは，1つだけで実行されたときの1/2の速度となる.

各ジョブは定まった時刻にシステムに提出される．しかし，この時刻は

あらかじめシステムにはわからない．システムには100Kバイトの利用可能な領域がある．実行すべき6つのジョブは次表のとおりである：

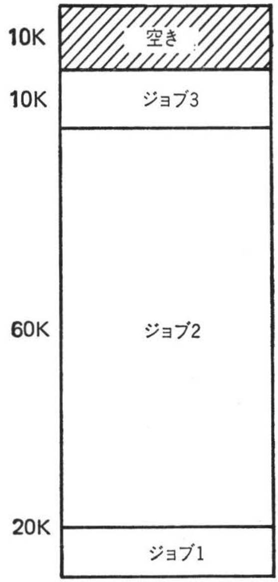
ジョブ番号	到着時刻	CPU時間	記憶容量	優先順位
1	0.0	1.0	20K	1
2	0.5	1.0	60K	2
3	1.0	0.5	10K	3
4	1.5	2.0	30K	3
5	2.0	0.2	20K	1
6	2.5	0.1	20K	2

システムによる処理を順を追って示すと次図のようになる：



ジョブ開始を（1）で囲み，終了を［1］で囲んで示している．かっこに囲まれた数字はジョブ処理に必要な残り時間を示す．

上の例では分割割付けを仮定している．したがって，時刻1.0では，記憶領域は次図のようになっている：



ジョブ・スケジューリングの効果を測定する1つの方法はジョブの返却時間を求めることである：

ジョブ番号	終了時刻 (F)	到着時刻 (A)	返却時間 (F-A)
1	1.77	0.0	1.77
2	2.85	0.5	2.35
3	2.4	1.0	1.4
4	4.85	1.5	3.35
5	2.6	2.0	.6
6	2.7	2.5	.2

平均返却時間
優先順位1（1および5）＝ 1.18時間
優先順位2（2および6）＝ 1.27時間
優先順位3（3および4）＝ 2.37時間

1. 再配置型分割割付けでは処理の順番がどのように変化するか簡潔に述べよ．平均の返却時間にはどのように影響するか．

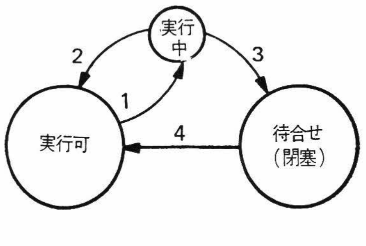
2. 再配置型分割方式による処理の軌跡と平均返却時間を求めよ（詰め合せ時間は無視できるとする）.
3. 問2の答えは問1で述べた予想に合うか．この予想は一般的に行なえるか．その理由を述べよ．
4. ジョブが多量の入出力を行なうとき，平均返却時間に対する影響はどうか．平均返却時間が影響を受ける理由は何か．

4.7 読者はオペレーティング・システムの設計者として，新システムのプロセスおよびジョブ・スケジューラの設計を任されている．高順位のジョブは可能な限り迅速に処理すること，およびシステム全体の仕事量とが重要な項目となっている．ジョブ・スケジューリングの方式としては，低順位のジョブより先に高順位のジョブをすべて処理することとする．プロセス・スケジューラの設計では次の2つの方式を考えている．第1は，ジョブ・スケジューラと同様の優先順位による方式である．第2は，利用者指定の優先順位とは無関係に，まず入出力限界のジョブを実行し，次に“普通”のジョブを，その後に計算限界のジョブを実行するものである．システム全体の目標に関して，この2つの方式を論じよ．

4.8

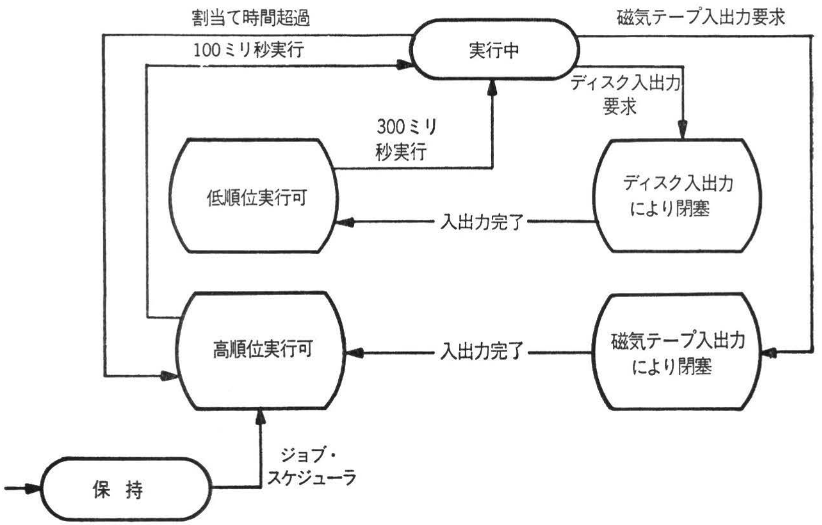
1. 詰め合せが CPU とは独立に入出力チャネルで実行できるとしたら（チャネル・コマンド語は，IBM 370 の MVCL 命令と同様に，番地 A から番地 B へ n バイトを転送することができる），どんな問題が残されているか．
2. チャネルが詰め合せを行なっている間に CPU で仕事をしようとする（多重プログラミング）．これを安全に行なうためのアルゴリズムを述べよ．流れ図を示してもよいし，処理の概要を述べてもよい．

4.9 次図のようなプロセスの状態遷移図を用いると，処理装置管理について多くのことを示すことができる：



1. どのような“事象”によって番号のつけられた遷移が行なわれるか.
2. システム内のプロセスをすべて眺めていると，1つのプロセスの状態遷移は他のプロセスの状態遷移をも引き起こすことがある．どのような条件の下で，1つのプロセスの遷移3が直ちに他のプロセスの遷移1を引き起こすことになるか.
3. 次に示す原因と結果の遷移が引き起こされるのはどのような条件の下でか.
   a．2→1
   b．3→2
   c．4→1
4. 次に示す遷移が他の遷移を引き起こさないのはどのような条件でか.
   a． 1
   b． 2
   c． 3
   d． 4

4.10 プロセス・スケジューラに対して次の状態遷移図を考えよう：



1. この図に示されているスケジューリングの原理を述べよ.
2. このスケジューリング方式を利用するのが好ましい条件を述べよ．ジ

ョブ・スケジューラは，このスケジューリング方式の下で実行するジョブを次のように3つ選択した：

ジョブ1　150ミリ秒計算して，磁気テープに要求を出し，150ミリ秒計算して，磁気テープに要求を出し，150ミリ秒計算して，磁気テープに要求を出す．

ジョブ2　250ミリ秒計算して，ディスクに要求を出し，50ミリ秒計算して，ディスクに要求を出す．

ジョブ3　250ミリ秒計算して，磁気テープに要求を出し，250ミリ秒計算して，磁気テープに要求を出す．

これらの3つのジョブは，システムに提出されるジョブの代表的な例である．磁気テープへの要求は500ミリ秒で完了し，ディスクへの要求は100ミリ秒で完了する．最初すべてのジョブは実行可であり，2つ以上のジョブが実行可状態のとき，スケジューラは，最も長時間実行可状態であったジョブを選択する(ラウンド・ロビン)．この3つのジョブが終了するまでは，ジョブ・スケジューラは他のジョブをアクティヴェイトしないと仮定してよい．

3. このシステムによってこれら3つのジョブを実行したときの記録を，次の書式によって，時間と遷移した状態を記入して作成せよ．

時刻	実行中	高順位 実行可	低順位 実行可	磁気テープ 入出力により 閉塞	ディスク入出力 により閉塞
t=0	1	2,3	—	—	—
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.

4. 各ジョブの返却時間と，CPU がアイドルになっている時間とを求めよ．

5. 問1で論じたような目的をこのプロセス・スケジューリングは果たしているか．いないとしたらその理由を示せ．

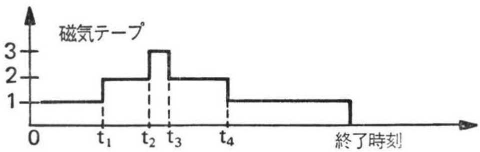
6. システムで実行するジョブの型が定まっているとして，一般にもっと能率よく動作させる（平均のジョブ時間とCPUアイドル時間，返却時間を減少させる）ためには，スケジューリングの方式をどのように変更したらよいか．

7. プロセス・スケジューラが問6の目的を達成するために，ジョブ・スケジューラはどのような方針でロードするジョブを選択すればよいか．

4.11 CPU とチャネルの間の処理装置の“競争”と“ロック”の問題は，2つの CPU の間の場合と同様か．異なるとすればどのような点が異なるか．

4.12 競争とすくみの相違点は何か．これらを防止したり軽減したりするにはどのようにすればよいか．

4.13 8台の磁気テープ装置を有する計算機システムを考える．このシステムで実行するジョブは次図に示したような利用の型に従って磁気テープを最大3台使用する．



t_i はジョブによって異なるが，いずれの場合にも時間 $t_2 \to t_3$ はきわめて短時間である．このシステムでのジョブ・スケジューリングは次のような種々の方法が考えられる：

1. ジョブが必要とする最大数の磁気テープ装置が空いているときに限りジョブを開始できる．このとき，これらの磁気テープ装置をジョブ処理の期間中割当てておく．
2. ジョブが磁気テープを要求すると残りの処理時間の間ずっと確保するものとして，すくみを防止する方法を用いる．
3. ジョブが磁気テープを解放し始めるとこれ以上磁気テープを要求しないということを知ったうえで，条件を緩和したすくみ防止の方法を用いる．

問 題

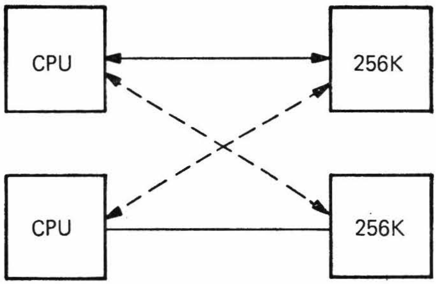
a．上述の3つの方法に対して流れ図を描け．

b．各方法について，多重プログラミングの最大多重度，および同時使用される磁気テープの最小数と最大数を求めよ．

4.14

1. IBM 360 の主記憶は，通常，記憶モジュールと呼ばれる256K バイト

の単位に分割されている．IBM 360/67 では，2つの CPU は記憶モジュールと組合せて全く異なった2種の構成にすることができる．第1，各 CPU はただ1台だけの記憶モジュールをアクセスでき，**2台の別個のシステム**がプログラムを処理することとなる．第2，両方の記憶モジュールは，1つのオペレーティング・システムの制御によって2台の CPU からアクセスできる．しかし，この構成のときは，通常，第1の構成のときより合計の仕事量が25%減少する．この性能低下は CPU 間の相互干渉のせいである．この相互干渉を起こす原因を示せ．



2. 相互干渉があっても，第2の構成を選ぶ理由は何か．
3. 第2の構成では，1つの処理装置がジョブ行列などのデータベースを変更している間，他の処理装置がそれをアクセスしないようにする必要がある．処理装置のロックアウトを行なう1つの方法は，ある処理装置がデータベースにアクセスしているか否かを示すバイトをデータベースに対応させることである．バイトの値が0のとき，データベースは空いておりアクセス可能である．値が0でないときは，このバイトを検査した処理装置はアクセスすることができず閉塞される．次に示すプログラムはこれを行なおうとするものである．正しく働くか調べよ：

```
         SSM    INTOFF      すべての入出力割込みを禁止する
         CLI    SWITCH,0    データベースに関するスイッチが0かどうか検査する

         BNE    BLOCKED     0でなければ，処理装置は別のプログラムに制御を
                            移すことができる

         OI     SWITCH,1    データベースのアクセスが閉ざされていることを
                            他の処理装置に通知

         SSM    INTON       再び入出力割込みを可能にする
         .
         .
         .
         NI     SWITCH,0    完了したときスイッチを0に設定する

         .
         .
         .
INTOFF   DC     X'0'
INTON    DC     (適当なマスク)
```

4. 検査設定命令は，処理装置ロックアウトを可能にするために特に設計されたものである．動作原理を説明せよ．

5. 並行的に再入可能でないプログラムの場合には，一時に1台の処理装置だけがこれを実行することができる．処理装置ロックアウトを行なうために，プログラムは次の形式をとる必要がある：

```
          L        1,BRANCH
LOOP      ST       1,LOOP

          順次再入可能コード

          L        1,STORE
          ST       1,LOOP
          .
          .
          .
STORE     ST       1,LOOP
BRANCH    B        LOOP
```

このプログラムの動作原理を説明せよ．この方法と検査設定命令の使用とを比較してみよ．

6. もし両方の CPU が，1台の記憶モジュールの番地の小さい同じ場所（たとえば，新 PSW，CAW）を使用したとしたらどのような問題が生じるか．もっとよい方法を述べてみよ．

4.15

1. すくみとは何か．
2. すくみの段階を設定して，これをどのようにしたら検出できるかを説明せよ（例は計算機科学に限定しなくてもよい）．
3. 記憶装置と磁気テープ装置などの入出力の両方のスケジューリングにおける一般的な原理はさほど差違のあるものではない．両方とも，資源の経済的かつ効果的な利用を達成するために，空いている資源を可能な限り多くの利用者に（多重化）最も効率よく割当てる問題を扱っている．この理由は，磁気テープやドラムなどは，主記憶と同様に，情報の記憶域の役割りを果たしているからである．主たる相違点は特定のデータを検索するアクセス時間にある．

a．本文では磁気テープ装置に関連してすくみの状態を論じた（図4.33など）．デマンド・ページングの割付けアルゴリズムの下で，主記憶が同様のすくみの状態となることが考えられるか．答えとその理由を述べよ．

b．Haberman などのすくみ防止技法は（第4.5.3.2参照）防止するだ

けに有効である．いったんすくみが起こると，それを解除する方法がないことが多い．上述の問 a に答えた後で，すくみがいったん起こったら解除することが不可能な理由を示せ．

4.16 総合的な計算能力を増大させるため，ある計算センターでは 2 台目の CPU を標準システムに追加して，両方の CPU で多重処理できるようにした．第 2 の処理装置はきわめて高価であるので，記憶装置は追加せず，両方の CPU は同一の主記憶を共用している．

1. 両方の CPU が番地の若い部分（PSW や CSW などが格納されている部分）を共用したらどのような問題が生じるか．もっとよい方法を考えてみよ．

2. 1 台の処理装置があるデータベースを変更している最中には他の処理装置がそれをアクセスしないようにする必要が時折起こる．これを**処理装置ロックアウト**と呼ぶ．どのようなデータベースに対して処理装置ロックアウトが必要か．どのようにしたら処理装置のロックアウトができるか．

3. 第 2 の CPU を追加した後で，合計の計算能力は 2 倍とはならず，ただの 1.6 倍に増加したことがわかった．この理由を説明せよ．

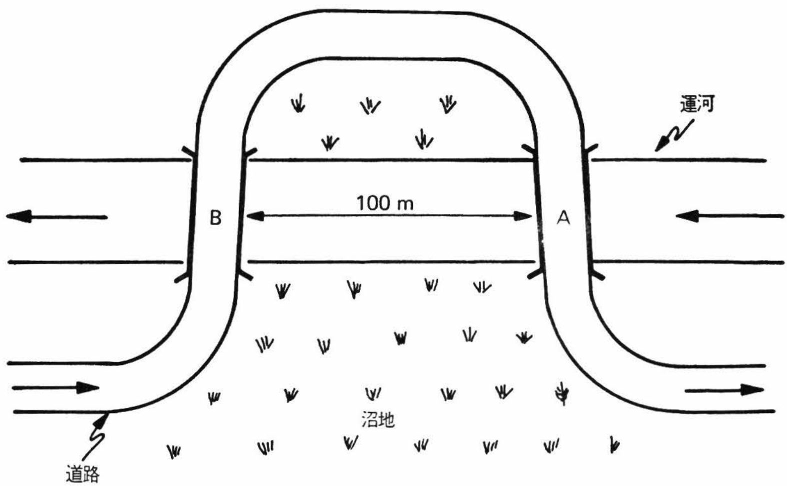
4.17 10台の磁気テープ装置を持つ計算機システムがある．このシステムで実行するジョブはすべて最大 4 台の磁気テープを要求する．しかし，ジョブは，長い間 3 台の磁気テープだけを使用して実行し，終了の真際に短時間の間 4 台目の磁気テープを要求することがわかっている．（ジョブは無限に提出されるとする）．

1. ジョブ・スケジューラが，空いている磁気テープが 4 台できるまではジョブを開始せず，磁気テープをジョブ処理の期間中割当てるという方針をとっているとすると，同時に実行できるジョブの最大数は幾らか．この方針の結果として実際にアイドルとなる磁気テープ装置の台数の最大値と最小値は幾らか．

2. 問 1 で述べたような過度に制限する方法の代わりに，すくみを防止するアルゴリズムが用いられるとする．同時に実行できるジョブの最大数は幾らか，この方針を用いることによって，実際にアイドルとなる磁気テープの最小数と最大数は幾らか．答えと理由とを述べよ．

3. すくみを防止して磁気テープを複数のジョブに割当てるアルゴリズムの流れ図を描け（問 2 のアルゴリズム）．

4.18　次図は，水門と2つの可動橋のある運河を示している．可動橋は沼地を避けるように運河を2回横切っている道路に設置されている．運河も道路も1方向だけの交通を通す．運河には主としてはしけが通る．はしけが橋に接近して橋から100メートルになると警報が鳴り，車が居ないとき橋は引揚げられる．橋は，はしけの船尾が橋を通過するまで引揚げられたままとなっている．同様な手順がB橋についても起こる．はしけの長さは普通200メートルである．この場合すくみが起こるか．その理由は何か．すくみが発生しようとしていることを検出するにはどのようにすればよいか．発生しかけているすくみを防止する方法を提案せよ．



4.19　プロセス間通信の例を4つあげよ．

4.20　実際のオペレーティング・システム（たとえば，第7章の例題のオペレーティング・システム）では4つ以上のプロセス状態を区別する必要がある．たとえば，プロセスが完全に生成されていないとき，プロセスはどの状態となっているか．また，ある事象が発生するまでスワップすることができないときプロセスはどの状態になっているか．

1.　実行可，実行，閉塞の3状態の図に少なくとも2つ以上の“実際の状態”を追加して，状態図を描き直せ．
2.　すべての遷移を示せ．
3.　新しい状態遷移を処理するのは，オペレーティング・システムのどの

モジュールか．

4. 問3のためのアルゴリズムを論じよ．

4.21 多重処理装置システムにおいてソフトウェア・ロックアウトのため停止しているプロセスと，多重プログラミング・システムにおいて閉塞されているプロセスの違いを述べよ．

4.22 入出力待合せ時間が25%であるとして，第4.2.4.3の例題のジョブの要求を解析してみよ．その結果を第4.2.4.2の結果と比較せよ．

4.23 第4.5.2.3節で述べた計数セマフォアを用いて，図 4.26 の受付け，店員，料理人，および会計の動作を定義せよ．結果は P_p および P_c のような形で示せ（第4.5.2.3節）．

4.24[1] ジョブ・スケジューラは，システムに到着したジョブをどのような順序で実行するかを決定する仕事を行なう．ジョブ・スケジューラは，ジョブ行列にジョブが入っていれば，どのジョブを開始するかを決定する．この問題では， M.I.T. の情報処理センターにおける追加の処理装置の中のジョブ・スケジューラのように，ジョブ・スケジューラの範囲に属する方針決定の処理を行なってみようとするものである．したがって，この問題を選択するならば，スケジューラは PL/I で作成しなければならない．

各ジョブをいつ実行するかを決定することのほかに，スケジューラは次のことを行なわなければならない：

1. 実行するために提出されたジョブの一覧表を編成し，維持しておくこと．
2. ジョブが終了したとき，一覧表の欄を抹消すること．
3. スケジューリングのためのプロセスが機能を発揮するように他のリストを維持したり変更したりすること．
4. スケジューリングの統計をとり，これらを用いて，性能をよくするよ

1) 本問はジョブ・スケジューラの設計と組込みの問題であり，流れ図や設計仕様は不要である．組込みにおいては次の内の１つのものを選択すること．

1. ジョブ・スケジューラの環境を模擬するシミュレータを用意する (Donovan, 1974)．
2. 自分で環境を整備する．この場合例題ジョブをスケジューラにデータとして与える．
3. 環境を模擬するシミュレータを自分で用意する．

計算機システムへの実際の組込みに対して用いられる付加的な設計の問題点については文献 *Pedagogical and Research Aids* (Donovan, 1974) を参照せよ．

うに動的にスケジューラを“調整”する．

システムには，ユーティリティ・プログラムによって，スケジューラに次の道具が用意されている：

1. 記憶管理
2. 装置管理
3. 時　計
4. ジョブが終了したときのシグナリング
5. 到着したジョブのジョブ制御情報をスケジューラに伝達すること．

本問でのスケジューリングには，磁気テープ以外の印刷機，穿孔機，ディスク，その他の装置要求を考慮しない．仮想的なシステムには磁気テープは10台あるものとする．記憶域は800Kバイトで，再配置できない区画として割付けられるので，断片化は重要な要素となる．しかし，ジョブ・スケジューラでは記憶域や磁気テープ装置を管理する必要はない．

スケジューラは，実際にはサブルーチンの集合で，スケジューリングの決定が必要なときにシステムから呼出される．これらのサブルーチンは次のように呼ばれている：

1. J_ARRIV　　ジョブ到着ルーチン
2. J_TERM　　ジョブ終了ルーチン
3. J_SCHED　　ジョブ開始ルーチン

以下にはこの機械向きの問題の目標を説明し，要求事項と任意選択事項とを示している．また，答えのルーチンとシミュレータとの関係，および結果として作成するデータの書式をも示している．

まず，ジョブ・スケジューラの入口の機能を述べる：

1. J_ARRIV　　システムが最初に呼出すルーチンである．このルーチンはジョブがシステムに到着する度に呼出される．システムは，到着したジョブに関する情報をすべて含む基底つき構造体のポインタを引渡す．この情報には，ジョブ番号，到着時刻，最大実行時間，必要記憶量，磁気テープ装置の必要台数，およびクラス（CPU 限界，平衡，入出力限界）が含まれる．J_ARRIV の仕事は，この情報を実行のために提出したジョブの一覧表に入れることである．
2. J_TERM　　ジョブが終了するたびに呼出されるルーチンである．終了すべきジョブのデータを納めた構造体へのポインタが渡される．スケ

ジューラの中の一覧表からこのジョブを追出すことが J_TERM の仕事である.

3. J_SCHED 上記以外のとき，すなわち，ジョブのスケジュールが可能と思われるとき呼出されるルーチンである. J_SCHED の仕事は次の事項の決定を行なうことである：

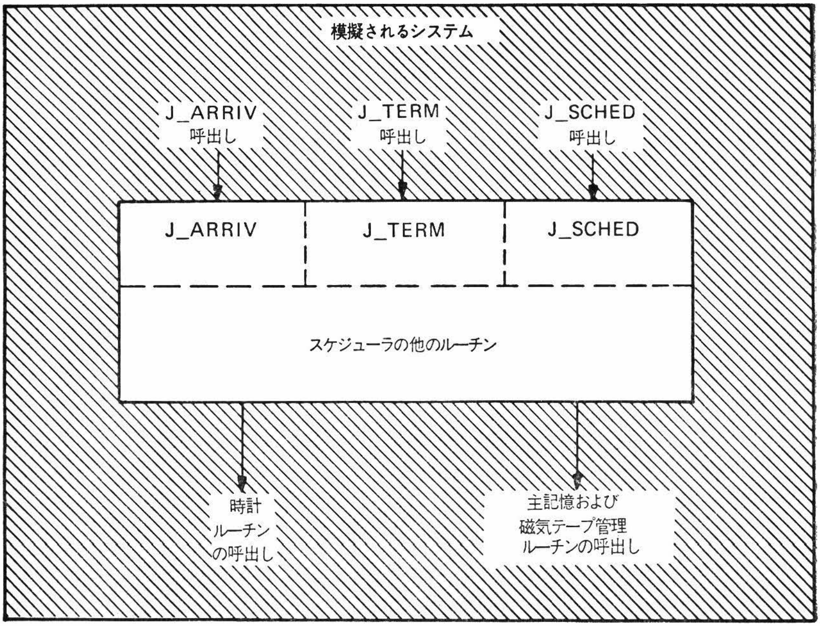
1. ジョブのスケジュールを行なうか否か.
2. どのジョブをスケジュールするか.

ジョブ・スケジューラが，このとき他のジョブをスケジュールしないならば JOB NUMBER=0 を返却する. ジョブをスケジュールしようとするときは，開始したいジョブの番号を JOB NUMBER の値として返却する. たとえば，大きなジョブが終了したときには，記憶域が再び一杯になるまで続けて J_SCHED が呼出されることがある. したがって，J_SCHED が一時にはただ1つの JOB NUMBER しか返却できなくても，実際上は制限事項とはならない.

次にスケジューラの主要な3つのサブルーチンに対するシステムの呼出しを示す：

1. J_ARRIV ジョブがシステムに到着したとき，ジョブをジョブ行列に加えるためにスケジューラに通知するために呼出される.
2. J_TERM ジョブが終了するとき呼出される.
3. J_SCHED ジョブをスケジュールするとき呼出され，ジョブが見つかればジョブ番号を返却する.

したがって，システムの構成と相互関係は次図のように示される：



本問では図の斜線の引いていない部分を PL/I で書くことである．

第Ⅰ部　次の因子を各ジョブごとに考慮してスケジューラを作れ：

1. 必要記憶量
2. 磁気テープ必要台数
3. ジョブの優先順位
4. 最大実行時間の指定
5. ジョブの到着時刻

第Ⅰ部では，すべてのジョブは同一のクラス（クラスB）に属し，クラスはスケジューリングのときに考慮されない．

模擬システムの統計を次に示す：

20%のジョブは 00.00 分実行する（すなわち，ジョブ制御言語の誤りな

どで実行されない).

1分間ジョブの50%は1/4分以下の時間実行する.

一般に，最大実行時間の指定 (MAXTIME) が大きいほど，その予測は正確である．ジョブの平均値としては，MAXTIME は10分，記憶域は150K，および磁気テープ装置2台である.

第II部　次のスケジューラは，ジョブ・クラス（計算限界，平衡，入出力限界）をスケジューリングで考慮するものである．さらに，最適なジョブ混合体を維持して，CPU の効率を最大とし，返却時間を最小にすることが要求される.

次表は，各種のジョブ・クラスの特性を要約したものである：

ジョブ種別	CPU時間	専有時間*
CPU制限	1	1.5
平衡	1	2.0
入出力制限	1	4.0

*専有時間はこのジョブをシステムで単独に実行したときの所要時間

上表の数値から，どのようなジョブ混合体を構成しても，多重プログラミングの場合のジョブの実行時間の合計は帯在時間の合計よりも小であることは明らかである．また，最適なジョブの混合体は，計算限界，平衡，および入出力限界のジョブを組合せてできることも明らかである．同時に実行するジョブの個数に対する最適なジョブの混合体の構成は公開問題となっているので，各自の実験では解かなくてもよい.

第III部　第I部と第II部に加えて，この部分では幾つかの任意選択機能を考える．それぞれの選択機能については，スケジューラは異なった方針によって動作する．次に示す目標の1つを組込め：

1. 最大仕事量　合計時間を最短としてすべてのジョブを実行する.
2. 返却時間　容量，装置の要求，あるいは優先順位に無関係に，ジョブの返却時間が60分を越してはならない.

III-1 あるいは III-2 を組込もうとするとき，全体的な返却時間や優先順位による返却時間を犠牲にすることは最小限でなければならない.

3. 急行処理　利用者がファイルを磁気テープ装置（システムには磁気テープが不足している）からディスク（ディスク装置が遊んでいる）へ切換えている間，運用の方針を決定する読者は，磁気テープ装置を使用しない低順位ジョブを，同じ料金のままで，一定限の時間に限って急行処理を行なうことにした．しかし，高い料金を支払っている高順位の利用者を混乱させないように次の3つの約束を行なったとする：

1　新しい急行処理の結果，高順位の返却時間が悪くならないこと．

2　磁気テープ装置を使用する高順位の利用者は少なくとも急行処理の利用者と同じ返却時間となること．

3　磁気テープ装置を使用しない高順位の利用者は最良の返却時間を得る．

低順位の急行処理をしない利用者には何も約束していない．

4. 面白くて意慾のあるスケジューリングの方針を構成して組込もうとするグループを歓迎する（教官と相談してから開始すること）．

第 5 章

装 置 管 理

本章では，処理装置および記憶装置以外の装置を，次に述べる理由により，ひとまとめにして論じる：

1. 各装置ごとに別々の章を設けるのは適当でない．
2. すべての装置に適用可能な一般原理を示すことができる．
3. 多くのシステムでは，これらの装置を管理するモジュールをすべて統一している．

本章では入出力装置の管理に焦点を合せて論じる．これらの入出力装置としては，実際の印刷機やカード読取り機，磁気テープ，ディスク，ドラムなどの入出力装置，および制御装置やチャネルなどの装置を含んでいる．入出力装置の特性については，第 5.2 節および第 5.3 節にて簡単に述べる．読者にはすでにこれらの装置を知っている人もあるであろう．これらの節では単に重要な装置の相対的な速度と特性を述べる．大規模な計算設備ではシステムの経費の半分以上を入出力装置に費やしている．したがって，これらの装置を有効に使用することが望まれる．

装置管理の基本的な機能は次のとおりである：

1. すべての装置の状態を把握する．このためには特別の機構を必要とする．普通に用いられる機構の 1 つは，各装置に関連する装置制御ブロック (UCB) のようなデータベースを使用することである．(UCB の書式の例について第 5.5.1 節を参照.)
2. どのプロセスが，いつ，どれだけの期間，装置を使用するかを決定する方針を定める．これらの方針を組込むためには広範な技法が利用できる．たとえば，装置の利用効率を高めようとする方針では，プロセスから出される量や速度にむらのある要求を多くの入出力装置に適する一定

速度の要求に整合させようとする．装置管理の方針を組込むためには基本的に3つの技法が存在する．

a．専　用　　装置は1つのプロセスに割当てられる．

b．共　用　　装置は多数のプロセスで共用される．

c．仮　想　　1つの物理的な装置は別の物理的な装置で模擬される．

3. 装置をプロセスに物理的に割付ける．同様にして，対応する制御装置やチャネルも割当てなければならない．

4. 装置を解放する．　解放は，プロセスの段階またはジョブの段階で行なわれる．ジョブの段階では，システムにジョブが存在する間装置が割当てられる．プロセスの段階では，プロセスが必要とする間装置が割当てられる．

本章では主として装置管理の方針に焦点を合せて論じる．その理由は，この技法は装置の特異性に依存する面が多く，ハードウェアの発達と共に急速に変化しているからである．しかし第7章では，各装置が単一のジョブに割当てられるような方針の組込みの例を詳細に述べる．

装置の状態を把握するモジュールは**入出力トラフィック制御**と呼ばれる．1台の装置の動作に関連するモジュールをすべて**入出力装置ハンドラ**と呼んでいる．図1.9に示したシステムの設計では，入出力装置は，その装置に関連する独立のプロセスに属する装置ハンドラを持っている．入出力装置ハンドラの機能は，必要な動作を行なうチャネル・プログラムを生成し，この装置の入出力を起動し，装置からの割込みを処理し，動作を最適化することである．一口でいうと，装置ハンドラは物理的な入出力を実行する．**入出力スケジューラ**は，要求に対していつ入出力処理装置を割当てるかを決定し，その装置に対する経路を設定する．

5.1 装置管理の技法

装置を管理し割付けるために次の3つの技法が使用される：

1. 専　用
2. 共　用
3. 仮　想

5.1.1 専用の装置

専用の装置は，ジョブが存在する期間中1つのジョブに割付けられる．装置によってはこの型の割付けが適している．たとえば，カード読取り機，磁気テープ装置，あるいは印刷機などを共用するのは困難である．幾人かの利用者が同時に印刷機を使用するとき，出力をはさみで切って糊で貼り合せなければならないであろうか．不幸にして，ジョブがその装置の全能力を連続的に利用しないならば，専用割付けは能率が悪い．したがって，適用可能なときは通常共用および仮想の技法が使用される．

5.1.2 共用の装置

ディスク，ドラム，その他大部分の直接アクセス記憶装置（DASD）などは，複数のプロセスによって同時に共用することができる．複数のプロセスは1台のディスクを，実質的には同時に，読出すことができる．共用装置の管理は，特に最高の効率を希望するときは，きわめて複雑となることがある．たとえば，2つのプロセスがディスクAを読出す要求を同時に出したとすると，どちらの要求を第1に処理するかを決定する機構が必要である．これは一部をソフトウェアで行なうことも（入出力スケジューラおよび入出力トラフィック制御），すべてハードウェアで行なうことも（きわめて複雑なチャネルや制御装置）可能である．

どのプロセスの要求を先に処理するかを定める方針は次に基づく：

1. 優先順位
2. システムの仕事を改善しようとすること（たとえば，ディスクの読出しヘッドに近い方の要求を先に選択する）．

共用の装置については，さらに第5.2.2.3節および第5.5節で論じる．

5.1.3 仮想の装置

普通なら専用されるものでも，装置によっては（たとえば，カード読取り機）スプーリングなどの技法を用いて共用の装置に変換することができる．たとえば，スプーリング・プログラムは，すべてのカード入力を高速でディスクに読込んだり，複写したりすることができる．後で，プロセスがカードを読込もうとすると，スプーリング・プログラムがその要求を中途でさえぎり，ディスクの読出しに変換する．ディスクは複数の利用者間で共用することが容易である

から，専用の装置を共用の装置に変換し，1台のカード読取り機を多数の“仮想的な”カード読取り機に変換したことになる．この技法は全く同じようにして，テレタイプ，印刷機，および大部分の専用的な低速入出力装置などの多くの種類の周辺装置に応用することが可能である．

第5.6節では，幾つかの装置を仮想化する技法と，記憶域に制限のあるシステムでこれらのアルゴリズムを組込む場合の問題点を論じる．

5.1.4 一般的な方策

一部の計算機の製造会社では本書で述べているよりも進んだ一般的な装置管理を行なおうとしている．たとえば，スプーリングの呼出し側の機能はファイル・システムと同様であり，バッファリングはスプーリングに著しく類似している．本書で述べているよりも進んだ一般化も可能で，記憶装置，処理装置および入出力装置は同じ原理を適用できる単純な装置とみなすこともできる．現在のところは，このような一般化は実際上制限があるけれども，将来はより一般的な原理が生み出され，すべての装置に適用可能な形で実際に応用されることになろう．したがって，このような一般的な原理を生む研究を続ける必要はあるが，一方，本書に述べた問題点を理解せずにはそのような原理は生まれ得ないであろうと考えている．

5.2 装置の特性——ハードウェアについて

およそ考え得るものはすべて計算機の周辺装置とすることができるし，実際にそうされている．これらは製鉄機械，レーザ光線，レーダ，光学機械，温度計，宇宙船など広い範囲にわたっている．幸いなことに，たいていの計算機設備は一部の周辺装置だけを使用している．周辺装置は大きく次の2つに分類することができる：

1. 入力また出力装置
2. 記憶装置

5.2.1 入力および出力装置

入力装置とは，計算機がこの装置を用いて外界を“感知”したり“触っ”たりするものである．これらの装置としては，温度計やレーダ装置のこともあるが，より一般的には，穿孔されたカードや紙テープを読取ったり，端末装置の

鍵盤から打鍵される情報を読取る装置である．**出力装置**とは，計算機がこの装置を用いて外界に“影響”を及ぼしたり，“制御”したりするものである．この装置としては，温度制御つまみやレーダの方向制御のこともあるが，より一般的には，カードや紙テープに穿孔したり，文字を紙の上に印刷したり，端末装置で印字を制御する装置である．

前章では印刷機やカード読取り機などの一般的な入出力装置について幾つか述べた(第2章バッファリング技法参照)．本章では**記憶装置**に焦点を合せて論じよう．

5.2.2 記憶装置

記憶装置は，計算機が情報を格納し（通常，**書込み**と呼ばれる手続き），後でその情報を取出す（**読出し**）ための装置である．原理的には，記憶装置は人間が用いる紙片と鉛筆と消しゴムの記憶機構と類似している．

記憶装置は次の3つの型に区分できる．この区分はアクセス時間（T_{ij}）の偏差の大小に基づいている：

T_{ij}＝最後にアクセスした項目を i として，項目 j をアクセスする時間

1. **順アクセス**　T_{ij} の偏差が大である（たとえば，磁気テープ)．
2. **完全な直接アクセス**　T_{ij} は一定である（たとえば，磁気コア)．
3. **直接アクセス**　T_{ij} の偏差はごくわずかである(たとえば，ドラム)．

5.2.2.1 順アクセス装置

順アクセス記憶装置は情報の位置づけとアクセスが物理的に順番となっている点に特徴がある．任意の項目をアクセスするには“逐次探索”を行なう必要があるので，平均アクセス時間は格納されている情報を半分だけ読出すのに必要な時間となる． 磁気テープ装置（MTU）は最も一般的な順アクセス記憶装置である．磁気テープ装置は，オーディオのテープ・デッキやカセット・テープと同じ原理に基づいているが，音楽や音声ではなく2進情報を記録する．典型的な順アクセス装置を 図 5.1 に示す．

通常，情報は，1バイトずつではなく，**レコード**と呼ばれるバイトの集合として格納される．一般にレコードは任意の長さ，たとえば1バイトから32,768バイト，とすることができる．図5.1では，すべてのレコード長は4バイトである．レコードは，標準のテープ・デックに録音された話し言葉と類似している．各レコードはテープ上の位置で，第1レコード，第2レコード，…という

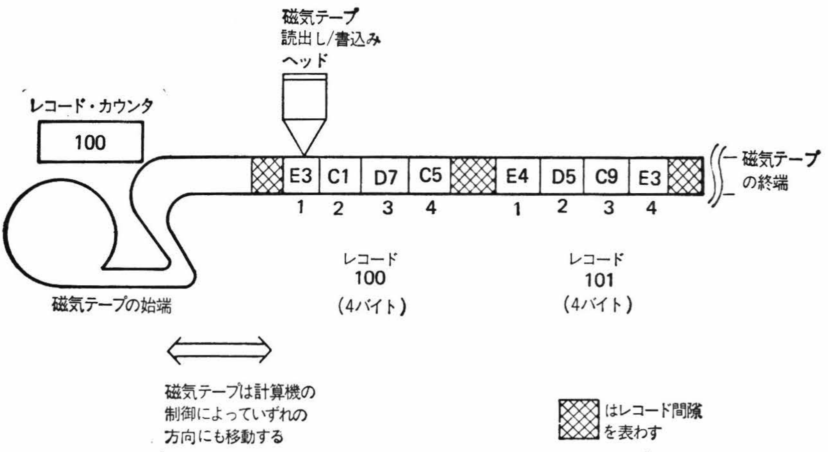


図 5.1 磁気テープ装置

ように識別することができる．図5.1のレコード・カウンタはオーディオ・テープ・デッキの“時間計”または“フィート計”と同様の機能を持っている．テープが始端に位置づけられており（完全に巻戻されている），レコード番号101のレコードを読みたいときには，それまでのレコード（1, 2, …, 99, 100）を読み飛ばさなければならない．

普通考えつく次のレコードの読出しや次のレコードの書込みなどの入出力コマンドのほかに，磁気テープ装置には，1つ前のレコードの読出し（逆読出し），テープの始端への高速巻戻し，読み書きしない1レコードの送りまたは戻しなどのコマンドがある．テープ上のレコード集合の区別を容易にするために，**テープ・マーク**と呼ばれる特別な型のレコードが存在する．この型のレコードはテープ・マーク書込みコマンドだけが書込むことができる．読出しは読飛ばしの動作中にテープ・マークが検出されると，特別の割込みが発生する．テープ・マークは，本のページを容易に探せるようにするしおりに類似している．

磁気テープを使用するための IBM 370 のチャネル・プログラムは次のようになる：

```
CCW   X'07',---,X'40',---       磁気テープを最初に巻戻す
CCW   X'37',---,X'40',---       レコード1を飛ばす
CCW   X'37',---,X'40',---       レコード2を飛ばす
CCW   X'01',DATA1,X'40',80      レコード3としてDATA1を書出す
CCW   X'01',DATA2,X'40',80      レコード4としてDATA2を書出す
CCW   X'1F',---,X'40',---       テープ・マークを書出す
CCW   X'07',---,X'00',---       再び磁気テープを最初に巻戻す
```

プログラムでは，一時に1つのレコードを読んだり書いたいするのに単純なチャネル・コマンド語1つからなるチャネル・プログラムを使用することができる．

磁気テープ装置は，幅1/2インチの磁気テープを使用し，1インチの長さに1600あるいは3200バイトを格納することができ，1秒間に最高200インチの速度で移動することができる．この速度をもってしても，標準の2,400フィートのテープを始端から始めて中央のレコードに達するには約1分かかる．

磁気テープ装置の典型的な特性は次のとおりである：

密度——1,600バイト／インチ（19,200バイト／フィート）
速度——200インチ／秒（$16\frac{2}{3}$フィート／秒）
長さ——2,400フィート（容量約46,000,000バイト）
最大アクセス時間——2分
平均アクセス時間——1分
順アクセス時間（次レコードの読出し）——4ミリ秒

T_{ij}の偏差の大きいことに注意せよ．

順アクセス記憶装置は，通常，パンチ・カードのデックの写しや，アセンブラまたはローダのパス1で作成された中間ファイルの写しなど，主として順番にアクセスする必要のあるような場合に使用される．今日のデータ処理では，大部分は基本的に順次的である．

図5.1に示したレコード間隙（IRG）は，テープを発進させたり停止させたりする物理的な制約から必要である．テープが停止した状態で次のレコードを読出すためには，規定の速度に達するまで一定の時間が必要である．同様に，レコードを読み終えた後に，停止するまでテープは若干走行する．

IRGは通常1/4インチから3/4インチである（400から1200バイト程度で

ある）．1つの装置での間隙の長さは一定である．このような理由で，1,000バイト以上ものレコードを格納することが普通行なわれる．さもなければ，テープの50%以上が無駄になる．たとえば，3/4インチの間隙をとって80バイトのレコードを格納すると，テープの約6.3%だけが有効に利用されるにすぎない（4,700万バイトのところが270万バイトなる）．

間隙による無駄を小さくするために，レコードをまとめて**ブロック**とすることが多い．**ブロッキング**とは，複数の論理レコードを1つの**物理レコード**（ブロック）にまとめることをいう．図5.2は，80バイトの論理レコード（カード）4つを1つの320バイトの物理レコード（ブロック）にブロッキングした例を示している．

普通，ブロックは800バイトから8,000バイトの長さである．もしブロックが長すぎると，信頼性に問題が生じ，読出したときに記憶域を取りすぎることにもなる．

ブロッキングとデブロッキングの機構は第6章で考察する．基本的には，ファイル・システムが1つの論理レコードを読出す要求を受付けると，この要求は適当な（ブロッキングされた）物理レコードの読出しに変換される．物理レコードの全体は主記憶中のバッファ領域に読出され，この領域から論理レコードを抽出することによって要求が満たされる．

バッファ領域の内容を管理する表を用いると，論理レコードのアクセスにおいて大幅な速度の改善を行なうことができる．たとえば，次の読出し要求は，通常，次の論理レコードに対するものであり，多分同一の物理レコードに入っている（現在バッファ領域に入っている）．したがって，磁気テープから物理レコードを再度読出す必要はない．論理レコードはバッファ領域から抽出することができる．

ブロッキングには主として3つの利点がある：

1. 各入出力動作によって一時に複数の論理レコードが読み書きされるので，入出力動作の回数が少なくてすむ．



図5.2 ブロッキングしたレコード

2. レコードの長さがレコード間隙より長いので無駄になる領域が少なくなる.
3. 無駄になっている部分が少ないので，多数のレコードを読出すとき，より少ない範囲を読むだけでよい（移動させる距離が小さい).

同時に次の3つの欠点がある：

1. ブロッキング，デブロッキング，およびレコードの管理のために，ソフトウェアのオーバヘッドがかかり，そのためのルーチンが必要である.
2. バッファ領域が無駄となる（たとえば，80バイトだけが必要であっても，8,000バイトを読出す必要がある). 小規模な計算機ではこの容量は全記憶容量の10%以上にもなることがある.
3. 長いレコードが読出されるのでテープ誤りの確率が増す.

5.2.2.2 完全に直接アクセスの装置

完全に直接アクセスの装置とは，アクセス時間 T_{ij} が一定であるものをいう. 磁気コア記憶装置，半導体記憶装置，読出し専用固定記憶装置，およびダイオード・マトリックス記憶装置などはすべて，完全に直接アクセスの記憶装置の例である.

図 5.3 は磁気コア記憶装置の動作を示している. 図の16個の円はフェライ

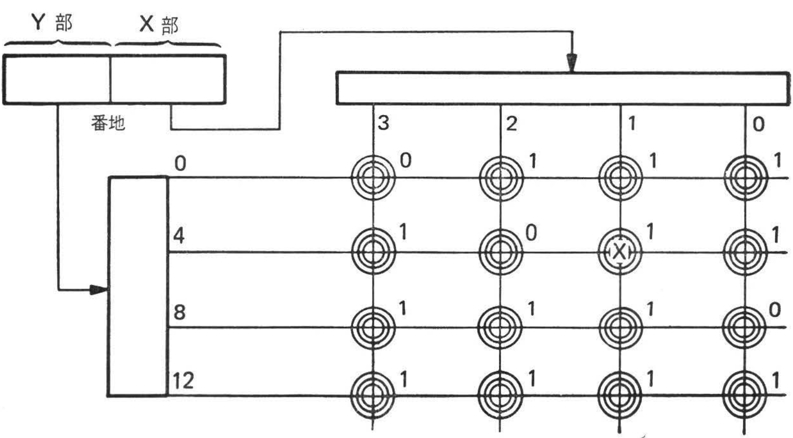


図 5.3 磁気コア記憶装置

トの磁気コアを表わしている．すべてのコアには2本ずつの選択線が貫通している．ハードウェアが1つの番地に対する読出し要求を受取ると，その番地はXおよびYの位置に分解される．次にハードウェアは適当なコアを選択する．この選択は2本の選択線に電流を通すことによって行なわれる．指定された座標（X，Y）の所に位置するコアだけに電流が“2重”に貫通し，その他のコアには電流は半分だけか，あるいは全く通電しないかである．2重に貫流する電流は磁場の極性を反転させる（1→0）のに十分な値になっている．

磁気コアの特性の1つは，磁束が変化すると電流を誘起することである．第3のセンス線がすべてのコアを貫通している．今目的のコアの状態が1であるとし，状態が0に切換わったとすると，センス線に電流が誘起する．状態がすでに0であれば，電流は誘起されない．

たとえば，図5.3で第5（2進数 0101）のビットを読出す要求が受付けられると，ハードウェアは番地を次のように分解する：

$$\underbrace{01}_{y}\underbrace{01}_{x} \quad \text{or} \quad \begin{array}{c} y=4 \\ \text{および} \\ x=1 \end{array}$$

したがって，センス出力は1となる（図 5.3 の×印のコア）．このコア読出しの過程で格納されていた情報が破壊されることに注意してほしい（すなわち，つねに0に設定される）．したがって，情報を元に戻しておくために余分のサイクルが必要である．

磁気コア記憶装置のアクセス時間 T_{ij} はすべての i および j について一定で，通常500ナノ秒から2マイクロ秒の範囲である．普通，この種の記憶装置は**主記憶装置**として使用される．場合によっては，価格が十分に低廉であれば（したがって普通低速で，2マイクロ秒以上である），記憶装置として使用される（たとえば，IBM の大容量コア記憶装置や CDC の拡張コア記憶装置）．完全な直接アクセスの装置をこのような方法で使用することは現在のところまれであるが，将来は多数使用されることになろう（Risko, 1968）．

最も新しい計算機では半導体記憶装置を使用している．基本原理は上述のコア記憶装置に類似しているが，半導体記憶装置は経費および速度の点で有利である．典型的な半導体記憶装置は50から500ナノ秒の範囲のアクセス時間で動作する．

この種の記憶装置はナノ秒の程度で共有できるけれども，一時には1つの番

地（あるいはブロック）を1台の CPU だけしか読出せないという制限がある． 2台以上の処理装置が記憶装置をアクセスすることがあるが，特に，1台の CPU と数台の入出力チャネルが接続されているとき，チャネルは**メモリ・サイクル・スチール**を行なっているということがある．メモリ・サイクル・スチールはシステムの性能を低下させることがあり，入出力トラフィック制御が処理しなければならない要素である．

5.2.2.3 直接アクセスの記憶装置（DASD）

直接アクセスの装置は，アクセス時間 T_{ij} の偏差の小さい装置である．この装置は**直接アクセス記憶装置**と呼ばれている．これらの装置は完全には直接アクセスできないが，この名前が使用されている．

本節では，2種類の直接アクセス記憶装置——固定ヘッドと可動ヘッドの装置——の例をあげる．

5.2.2.3.1 固定ヘッドのドラムとディスク

図 5.4 は磁気ドラムと同様の直接アクセスの記憶装置を示している．

磁気ドラムは，簡単には短い磁気テープ片の始動と終端とを接続して，同筒上に並べて置いたものと見なすことができる．各テープ片は**トラック**と呼ばれ，独立の読出し／書込みヘッドを持っている．ドラムは高速で回転し続けており，レコードは高速で読出し／書込みヘッドの下を繰返し通過する．各レコードはトラック番号とレコード番号の組で識別することができる．たとえば，図 5.4 でレコード（2, 1）は X 'C4C9E2D2' であり，レコード（5, 1）は X 'D1D6C8D5' である．普通の磁気ドラムは高速で回転しており，数 100 個の読出し／書込みヘッドを持っている．したがって，磁気テープ1巻に入っているレコードを乱アクセスするのに平均1分かかるのに対して，ドラムを乱アクセスして読み書きするのは5ミリ秒から10ミリ秒で完了する．

直前あるいは直後のレコードしか直接アクセスできない磁気テープ装置と異なり，直接アクセス記憶装置は任意のレコードをアクセスすることができる．したがって，入出力チャネル・コマンドの中に直接アクセス記憶装置の番地を指定することが必要である．論理的には入出力コマンドは次の項目から構成される：

1. 直接アクセス記憶装置のレコード（2, 1）を，
2. 主記憶番地（14680）へ，
3. 長さ（80）

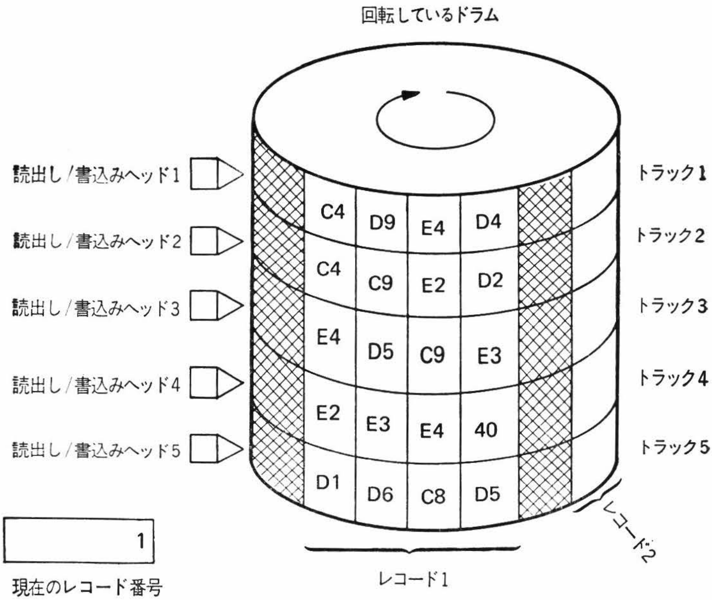


図 5.4 固定ヘッドのドラム (DASD)

を読込め.

多くの場合この要求は次の2つの部分に分割される：

1. 直接アクセス記憶装置の位置決め動作
2. データ転送動作

したがって，上述の入出力コマンドは次のようになる：

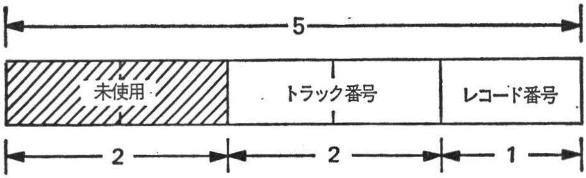
直接アクセス記憶装置のレコード (2, 1) への位置決めをする.
主記憶番地 (14680) へ長さ (80) だけ読込む.

位置決めがなされると，直接アクセス記憶装置は順アクセス装置と見なすことができる. レコード (2, 1), (2, 2), および (2, 3) を読出そうとするとき，入出力コマンドは次のようになる：

直接アクセス記憶装置をレコード (2, 1) に位置決めする.

主記憶番地（14680）に長さ（80）を読込む.
主記憶番地（14760）に長さ（80）を読込む.
主記憶番地（14840）に長さ（80）を読込む.

IBM 370 においては，直接アクセス記憶装置のレコード番地は次のように5バイトで表わされる：



“未使用”とされる部分については後で説明する．IBM 370 での位置決め動作は，実際には次の2つのチャネル・コマンドによって行なわれる：

```
GET     CCW X'31',ADDR,X'40',5        IDの探索：これは正しいレコードか
                                               正しい場合には次のCCW を飛ばす

        CCW X'08',GET,X'40',--        制御の移行：正しくなければ次のレコードを調べる

        CCW X'06',DATA,X'00',80       DATAへ80バイト読込む
ADDR    DC   X'0000000201'            トラック2，レコード1
DATA    DS   CL80                     80バイトのデータ領域
```

等しい識別番号を探索するコマンドは，現在読み書きヘッドの所にあるレコード（あるいは，レコードの中途のときは次に到着するレコード）の番地を比較する．このレコードが所要のものであれば，次のチャネル・コマンド語は飛越される．異なっていれば，次のチャネル・コマンド語が実行される．直後のコマンド語が探索コマンドへの制御移行コマンドであれば，所要のレコードが読み書きヘッドの所に到着するまで各レコードを検査するループが形成される．

ドラムの典型的な特性は次のとおりである：

回転速度＝10ミリ秒
最大アクセス時間＝10ミリ秒

平均（乱）アクセス時間＝5ミリ秒
順アクセス時間（レコードの長さに関係する）＜1ミリ秒
T_{ij} の偏差は小さい（1桁の差）．
容量＝800万バイト（256ヘッド＊32,000バイト／トラック）

ドラムのようにつねに回転している装置と，磁気テープのように起動／停止を行なう装置の間には幾つか大きな相違点が存在する．ドラムでは回転が止まることがないので，レコード間隙（IRG）は不要である（磁気テープでは起動/停止のために IRG を必要とした）．ドラムで IRG を利用するのは，1つはドラムのレコードを順に読出すとき CPU で計算する時間を作り出すためである．たとえば，IRG を通過する時間が最後のレコードの処理と次のレコードの入出力を開始させる時間にほぼ等しくしておけばよい．このようにしておかなければ，次のレコードを得るには，ドラムが完全に1回転するのを待たねばならない．これでは順次処理において時間の差がきわめて大きくなる（1ミリ秒のアクセスが10ミリ秒のアクセスとなる）．別の方法は，レコードを格納するのに“石けり遊び”の模様を使用することである．たとえば，10個の物理レコード（A，B，C，…，J）があり，トラック当たり10レコードのドラムに格納するとするとき，次のようにして格納することも可能である：

ドラムのレコード	データ・レコード
1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H
9	I
10	J

これらのレコードがしばしば順番に読出されると仮定しよう（ジョブはレコードA，次にB，……を要求する）．ジョブで1つのレコードを処理するのに読出し後2ミリ秒かかったとすると，レコードA（第1レコード）を読出して処理した後にドラムは第4レコードの始点まで回転している（1回転10ミリ秒，すなわち，1レコード1ミリ秒とする）．レコードBを読出すためには，ドラムは再びレコード2の所へ回転してこなければならない．これには8ミリ秒かかる．上の代わりに，レコードを次のように格納することもできる：

ドラムのレコード	データ・レコード
1	A
2	H
3	E
4	B
5	I
6	F
7	C
8	J
9	G
10	D

この場合でも，レコードAを読出して処理をすませた後に，ドラムは第4レコードの始点に回転してきている．次にレコードBが必要であるが，これはちょうど第4レコードに格納されている．したがって，余分に回転を待合せる必要はない．データ・レコードを格納するための2つの方式を比較して，最初の方式では10レコードすべてを処理するのに大略次の時間がかかる：

10×[1ミリ秒（レコード読出し)+2ミリ秒（レコード処理)+8ミリ秒(次レコードのアクセス)]＝10×11ミリ秒＝110ミリ秒

第2の方式では“次レコードのアクセス”が除去され，時間は次のようになる．

10×[1ミリ秒（レコード読出し)+2ミリ秒（レコード処理)]
＝10×3ミリ秒＝30ミリ秒

いい換えると，第2の方式はおよそ4倍速い．この差は，1,000レコードとか10,000レコードの処理を行なうときにはきわめて重大である．

順次処理の能率を改善するには，各ドラム・レコードの中で2つ以上の連続したデータ・レコードがアクセスされるので，ブロッキングが有効である．ドラムで IRG を使用している場合，ブロッキングは IRG による無駄な領域を減少させる効果がある．

5.2.2.3.2 可動ヘッドのディスクとドラム

磁気ディスクは磁気ドラムと類似した装置であるが，1枚以上の円板を用い，各円板は同心円状のトラックを持ち，円板ごとに1個の読出し／書込みヘッドがある．これは蓄音機のディスクと同様である．読出し／書込みヘッドは高価であるから，装置によってはトラックごとにヘッドを持たず，ヘッドが1つのトラックから別のトラックへ物理的に移動する．この種の装置は，**可動腕**あるいは**可動ヘッド型直接アクセス記憶装置**と呼ばれる．腕の各位置はシリン

ダと呼ばれる.

図5.5に示したような可動ヘッド型直接アクセス記憶装置のレコードを識別するには，腕の位置，トラック番号，およびレコード番号を指定する必要がある．腕の位置は半径方向の変位であり，レコード番号は円周方向の変位である．したがって，レコードにアクセスするには，ヘッドがまだ正しい位置になければ半径方向に移動して，次にレコードが読出し／書込みヘッドの所へ回転してくるのを待合せなければならない.

IBM 370 では，腕の位置決めのために**シーク**という独立の入出力コマンドが用意されている．腕の位置決めが行なわれた後では，この装置は第5.2.2.3.1節で述べた磁気ドラムと全く等しくなる．ディスクでのレコードは次のように6バイトを使用して指定される：

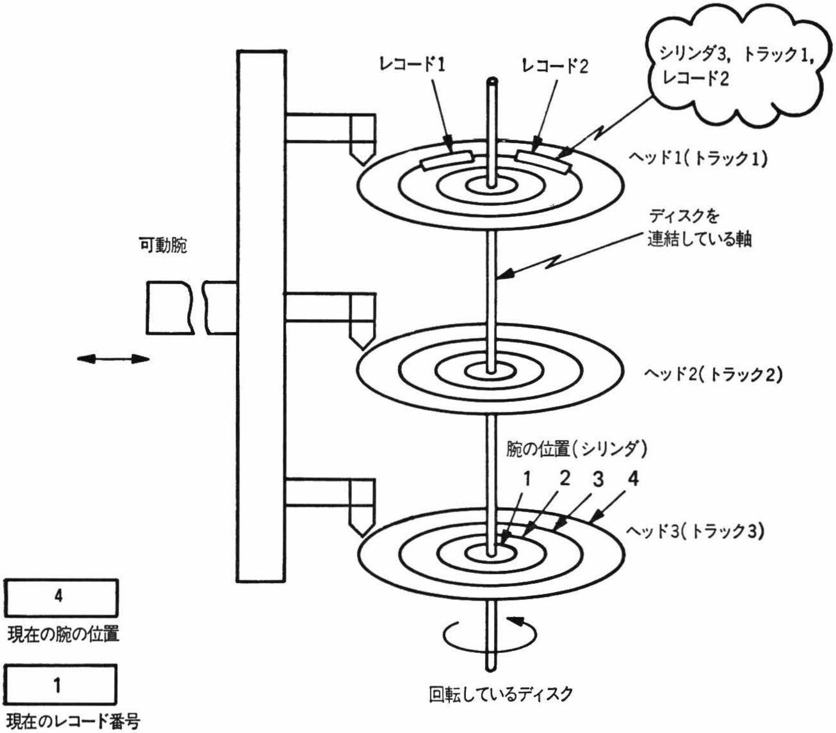
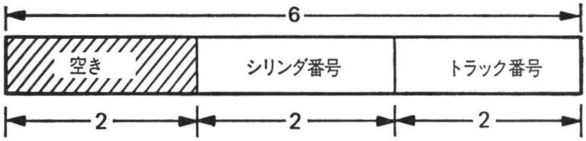


図 5.5 可動腕ディスク



識別番号探索コマンドを用いてレコードを探索するのとは異なり，腕が一度あるシリンダに位置決めされると，別のシーク・コマンドが出されるまで，そのシリンダに留まっている．このように，シーク・コマンドは予備シークと呼ばれて，別個に実行されたり，長いチャネル・プログラムの一部として実行されたりする．

ヘッドが現在シリンダ3に位置決めされていないと仮定して，シリンダ3，トラック1，レコード2のレコードを読込むためのコマンドを 図 5.6 に示す．

通常の可動腕型ディスクは，10～50トラック，200～400シリンダを持ち，次のような特性を持っている：

最大アクセス時間＝75ミリ秒
（腕移動の最大時間55ミリ秒，回転時間1周20ミリ秒）
平均（乱）アクセス時間＝30ミリ秒
順アクセス時間＜1ミリ秒
　T_{ij} の偏差は小さい（2桁以下の差である）
容量＝1億バイト（400シリンダ＊20ヘッド＊16,000バイト）

5.2.2.3.3 一般的な特性

直接アクセス記憶装置によっては各レコードにキーをつけることができるものもある．キーは利用者によって指定され，レコードの内容を示すようにする

```
        CCW  X'0B',ADDR,X'40',6           シリンダをシークする
SEARCH  CCW  X'31',ADDR+2,X'40',5         所要のレコードを探索する
        CCW  X'08',SEARCH,X'40',--        探索を繰返す
        CCW  X'06',DATA,X'00',80          DATAへ80バイトを読込む
ADDR    DC   X'00 00 00 03 00 01 02'      完全な番地：シリンダ1,
                   シリンダ トラック レコード   トラック1，レコード2
DATA    DS   CL80
```

図 5.6 レコードを読出す CCW

ことができる．この種の装置の制御装置では，トラックとシリンダの指定ではなく，キーの指定でレコードを読み書きするコマンドを用いることができる．たとえば，

'識別番号＝0201を探索せよ'とする代わりに
'キー＝HARRY を探索せよ'とする．

この形式の番地指定ができる直接アクセス記憶装置には，たとえば，読出しヘッドは正しいシリンダに位置決めされている必要があるといった条件が課せられることがある．

CPU の価格が逓減する傾向があるので，ディスクでレコードを探索せずに主記憶中に表を作っておき，チャネルの経費の減少をはかった方がよいこともある．

以上は記憶装置の代表的な例である．多数の変形があるが，すべては同様な特性を持っている．

5.3 チャネルと制御装置

装置管理は，チャネルや制御装置の制御も行なう．第2章で説明したように，**チャネル**はチャネル・プログラムを実行する特別の処理装置である．チャネルは高価であるから，装置の台数よりもチャネルの台数の方が少ない．したがって，チャネルを1つの装置から別の装置へと切換える必要がある．同時に動作する入出力装置の台数はむしろ少ないので一層価格を逓減することもできる．したがって，装置の電子回路と機械部分とを分割することが賢明である．数台分の電子回路は1台の制御装置として，装置間で切換えて使用することができる．図5.7は，チャネル，制御装置および装置の関係を示している．普通は，1台のチャネルに最大8台の制御装置が，1台の制御装置に最大8台の装置が接続できる．

入出力動作を起動させるためには，主記憶と装置との間に情報の経路が必要である．たとえば，ドラムJをアクセスするには，チャネルAとドラム制御装置Dとを使用する必要がある．ドラムGへの入出力動作が進行中であれば，チャネルAを使用できないので，ドラムJの入出力を起動することができない（ドラムGの入出力には，チャネルAと制御装置Cとを使用する必要がある）．

装置管理ルーチンは，

1. チャネル，

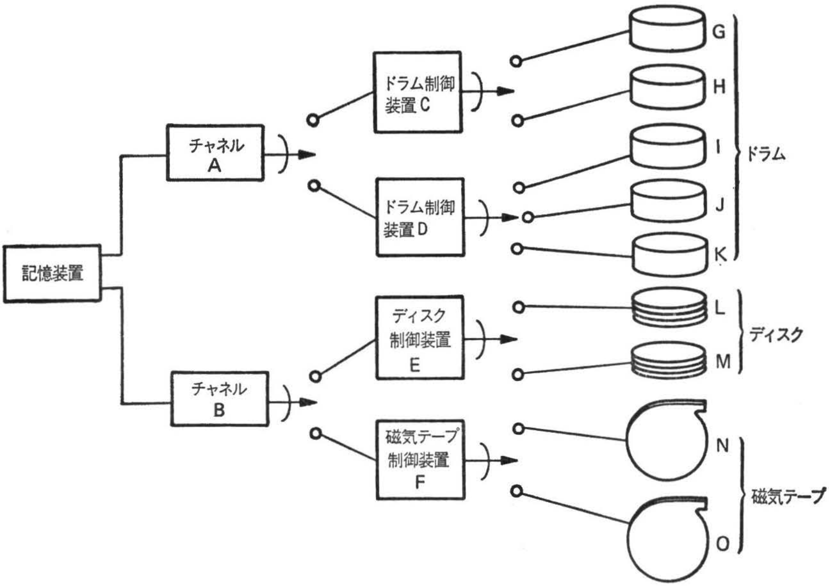


図 5.7 計算機システムにおける入出力の構成

2. 制御装置，および
3. 装 置

の状態を把握していなければならない．チャネルおよび制御装置の台数は通常少数であるので，しばしば大きなあい路となることがある．このために，装置の独立な動作，装置でのバッファリング，多数の経路，およびブロックの多重化などの技法を用いることができる．

5.3.1 装置の独立な動作

多くの場合，装置にはある程度のチャネルや制御装置の助けだけで，入出力動作を完了できる機能がある．ディスクのシーク動作は50ミリ秒かかるもかもしれないが，チャネルがシーク番地を主記憶からディスクに伝達するためには10〜20マイクロ秒あれば十分である．シークは装置単独で完了することができる．同様にして，装置と制御装置だけで，チャネルを用いず完了することので

きる動作もある.

入出力動作を起動するときには，経路は完全に空いていなければならない．経路の部分が空いたことを通知するために，チャネル完了，制御装置完了，装置完了という別々の入出力完了割込みが用いられる．入出力完了割込みはチャネル状態語の状態ビットによって指示される．経路の中の2つ以上の部分が同時に空いたときには，割込みは1回だけであるが，状態ビットは空いた部分をすべて指示する（たとえば，チャネル完了と制御装置完了の2つのビットがチャネル状態語の中でオンとなる).

チャネル完了が発生すると，装置はまだ動作中であっても，チャネルの動作は終了する．このとき，そのチャネルを用いて，別の装置の入出力動作を起動することができる.

5.3.2 バッファリング

装置や制御装置の内部にデータ・バッファを設けることによって，普通なら長時間チャネルを必要とする装置も，チャネルと独立に動作を遂行することができる．たとえば，カード読取り機は1枚のカードを読取りチャネルを通じてデータを転送するのに物理的には約60ミリ秒かかる．しかし，**バッファつきカード読取り機**は，カード読取り機あるいは制御装置の中のバッファにあらかじめ1枚のカードを読込み，80バイトの情報を格納している．チャネルがカードの読込みを要求したとき，バッファの中の80バイトの内容は高速（たとえば，100マイクロ秒）で主記憶に転送され，チャネルは解放される．その後装置は，独立に次のカードの読込みを行ないバッファの中に入れる．カード読取り機，カード穿孔機，および印刷機にはバッファが設けられることが多い.

5.3.3 多重の経路

チャネルと制御装置におけるあい路を解消するために，より多くのチャネルと制御装置を設置することもできる．しかし，より経済的な方法は，チャネルと制御制置をもっと融通のきくように，各装置に複数の経路を可能とすることである．図5.8はこのような入出力装置の構成を示している．この例では，装置Eに対して4つの経路が存在する：

1. チャネルA——制御装置C——装置E
2. チャネルA——制御装置D——装置E
3. チャネルB——制御装置C——装置E

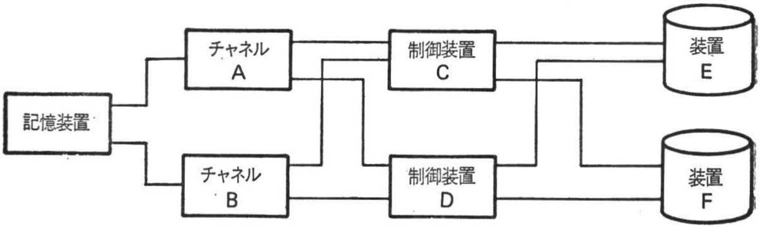


図 5.8 複数の経路を有する入出力構成

4. チャネルB——制御装置D——装置E

装置Fの入出力のためにチャネルAおよび制御装置Cが使用中であれば，装置Eを動作させるためにはチャネルBおよび制御装置Dの経路を見つけることができる．実際の入出力装置の構成は，図5.8のような台数の少ないものではなく，チャネル1台に8台の制御装置と，制御装置1台に8台の装置を接続することができる．完全な構成では，制御装置はどのチャネルにも接続され，装置もどの制御装置にも接続され，主記憶と各装置の間には64とおりの経路が存在する．このような極端な構成は，望ましいけれども，経済的な理由によってまず行なわれることはない．

融通性を与えることのほかに，複数の経路を持つ入出力の構成は信頼性の面でも都合がよい．たとえば，チャネルAが故障して修理する必要があっても，すべての制御装置は依然としてチャネルBを通じてアクセスすることができる．入出力の構成におけるすべての経路の状態情報を管理し，要求された装置にアクセスするために使用できる経路を見つけることは，装置管理の責任である．

5.3.4 ブロックの多重化

通常の IBM 370 のセレクタ・チャネルでは，装置の独立な動作とバッファリングは，1つのチャネル・プログラムのチャネル・コマンド語だけにしか適用できない．チャネルと制御装置は，チャネル・プログラムの実行中は接続されたままになっている． IBM 370 の**ブロック・マルチプレクサ・チャネル**は複数のチャネル・プログラムに同時にサービスすることができる（8つのチャネル・プログラム）．バッファリングされていたり装置の独立の動作などでチャネルをしばらく必要としないような入出力動作を行なうことになったときに

は，チャネルがサービスを要求している別のチャネル・プログラムに自動的に切換わる．ブロック・マルチプレクサ・チャネルはチャネル・プログラムに対する多重プログラミングのハードウェアによる組込みである．

この型のチャネルの多重プログラミングは，チャネル・プログラムすべてを1つの長いコマンドとすることによって，セレクタ・チャネルで模擬することができる．このようにすると，装置管理のソフトウェア・ルーチンによって，必要に応じてチャネルを切換えることができる．しかし，チャネルの切換えは頻繁に，しかも迅速に行なわねばならないので，中央処理装置が極度に高速でなければあまり魅力的なものではない．

5.4 装置の割付け

本章の始めに述べたように，装置の割付けには，専用，共用，あるいは仮想の3つの方法が存在する．第5.6節で述べる仮想割付けは，専用装置を共用装置に変換する．したがって，この方法は専用および共用割付けの特別な場合に過ぎない．

装置によっては，ジョブで使用する前に，カード読取り機のホッパーにカード・デックを人手で置いたり，必要とするレコードのところに磁気テープを位置決めしたりするような人間の動作や時間のかかる入出力動作が必要である．このような装置を複数のジョブの間で切換えるには，余分の仕事が必要となる．したがって，これらの装置は一時には1つのジョブに専用使用され，ジョブの完了の時点でジョブがこの装置を解放するまで他に割付けられない．装置をジョブに割付けることは，通常，処理装置管理のジョブ・スケジューリング・ルーチンと関連して行なわれる．

磁気ドラムやディスクなどの直接アクセス装置は，容易にしかも高速に位置決めすることができるので，この装置を並行して実行している複数のジョブの間で共用することが可能である．共用装置は，論理的には，専用装置群と見なして取扱うことができる．共用装置を使用しているジョブには論理的に別個の領域が割当てられている．この装置は直接アクセスの特性を持っているので，これらの領域をジョブの間で切換えることが経済的に可能となる．装置管理ルーチンは，ジョブにどの共用装置のどの領域を割付けるかを決定する．この決定はしばしばジョブ・スケジューリング・ルーチンと関連しても行なわれる．

専用および共用装置のどちらに対しても，静的および動的な割付け決定を行なう必要がある．静的な割付けは，主として，どの専用および共用装置をジョ

ブに割当てるかを決定するものである．一般に，装置管理においては，チャネルと制御装置のあい路を最小にするために，広い範囲の装置に割付けを行なう．多くの場合，利用者はジョブ制御カードを用いて，どの装置を同時に使用し（別々のチャネルに接続されているべきである），どの装置を別々に使用するか（これらは同じチャネルに接続されてもよい）をシステムに通知することができる．IBM の OS/360 のジョブ制御言語の選択機能である同種（AFF）と異種（SEP）はこの目的を果たすものである（Brown, 1970；Cadow, 1970）．

チャネルと制御装置の台数は少ないので，これらが恒久的にジョブに割当てられることはまずない．割当てはジョブが装置管理に対して入出力要求を発したときになされる．割当てられたチャネルや制御装置は，入出力要求が完了すると直ちに解放され，他に使用することができるようになる．

装置管理の静的な割付けは，処理装置管理におけるジョブ・スケジューラの静的な決定と似ている．動的な割付けの目的は，処理装置管理におけるプロセス・スケジューラと類似している．本章の残りの部分は，主として装置管理の動的な割付け機能について論じる．

5.5 入出力トラフィック制御，入出力スケジューラ，入出力装置ハンドラ

装置管理の機能は便宜上次の 3 つの部分に分けることができる：

1. **入出力トラフィック制御**　すべての装置，制御装置，およびチャネルの状態を把握する．
2. **入出力スケジューラ**　ジョブからの入出力要求によるアクセスに対して，チャネル，制御装置，および装置を割付けるための方針を組込む．
3. **入出力装置ハンドラ**　入出力スケジューラが決定を行なった後，チャネル・プログラムを生成し，入出力起動命令を発し，装置割込みの処理をして，実際の動的な割付けを遂行する．通常，装置の種類ごとに別個の装置ハンドラが設けられる．

5.5.1 入出力トラフィック制御

入出力スケジューラが，主として**方針**（誰が，いつ装置を使用するか）に関係するのに対して，入出力トラフィック制御は**動作**（装置を割当てることができるか）に関係する．トラフィック制御はすべての状態情報を管理している．

入出力トラフィック制御は，チャネルや制御装置の台数が少なく，図5.7および5.8に示したような複数の経路の問題が存在するので，複雑になる．トラフィック制御は最小限次の3つの問に対して答える：

1. 入出力要求にサービスするための経路が空いているか．
2. 2つ以上の経路が空いているか．
3. 経路が空いていないときには，いつ空くか．

これらの問に答えるために，トラフィック制御は装置の状態と接続関係を示したデータベースを管理している．これは図5.9に示したような装置制御ブロック(UCB)，制御装置制御ブロック(CUCB)，およびチャネル制御ブロック(CCB)を用いて行なわれる．第1の問は，要求の装置の制御ブロックから開始して，接続されている制御装置およびチャネルを調べ，空いている装置の組合せを求めることにより答えることができる．同様にして，空いている経路をすべて求めることができる．第2の問は，図5.10に示したように入出力の構成が対称でないときには，1つの経路を選択したとき他の入出力要求を閉塞

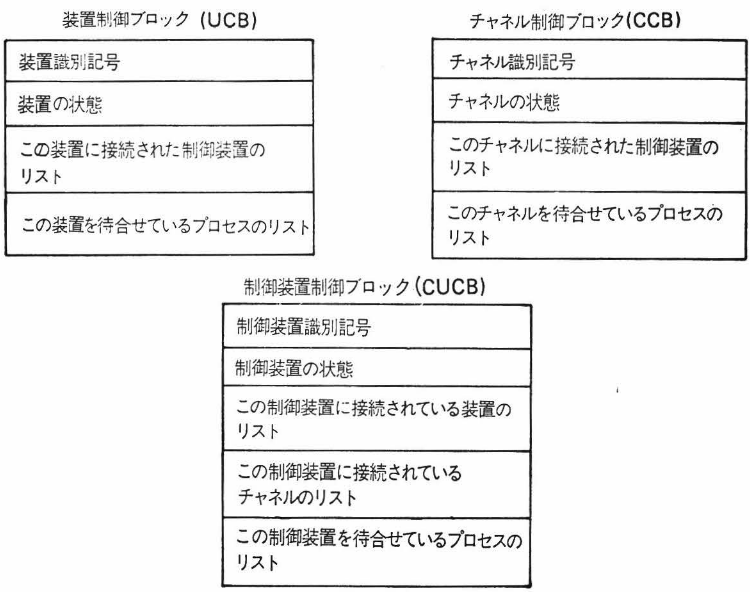


図5.9　制御ブロック

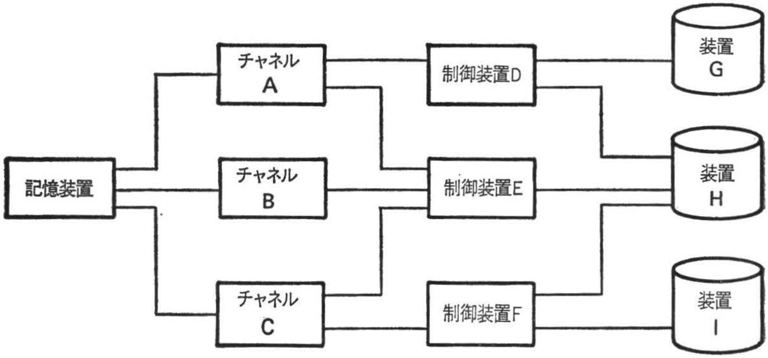


図 5.10 非対称な入出力の構成

してしまうことがあるので，特に重要となる．この場合のように，ある経路は他の経路よりも入出力要求を閉塞することが少ないことがある．図 5.10 では，装置Hには5とおりの経路が存在するが，装置GまたはIにはただ1つの経路しかない．

入出力の負荷が大きい場合には，プロセスから入出力要求が発せられたときに空いている経路が存在しないことがある．入出力完了の割込みが発生すると，1つ以上の要素（装置，制御装置，あるいはチャネル）が再び利用できるようになる．このとき，待合せている入出力要求のすべてについて，空いた経路を使用して入出力が可能かを調べることができる．しかし，多数の入出力要求が待合せているならば，この検査は時間がかかる．もっと能率のよい方法は，図 5.9 に示したように，1つの要求に“関係”する入出力要求のすべてをその装置の制御ブロックの中に一覧表としておくことである．このときには，特定の入出力要求のみを検査すればよい．

5.5.2 入出力スケジューラ

使用できる経路の数よりも多くの入出力要求が保留されているときには，どの入出力要求を最初に実行するかを選択する必要がある．プロセス・スケジューリングで述べたことの多くがこの場合にも適用できる．1つの大きな相違点は，通常，入出力要求の処理には割当て時間というものがないということである．すなわち，いったんチャネル・プログラムの実行を開始すると，動作を完了するまで中断されることがない．たいていのチャネル・プログラムはきわめ

て短かく50〜100ミリ秒で終了する．しかし，数秒から数分の時間かかって終了するチャネル・プログラムも存在する．

入出力スケジューラには各種の方針を用いることができる．たとえば，プロセス・スケジューラがプロセスに高い優先順位を割当てているならば，入出力要求に対しても高い優先順位を割当てることが適当である．このようにすると，そのジョブを可能な限り迅速に終了させることができる．

入出力スケジューラが入出力要求の相対的な順番を決定すると，入出力トラフィック制御は，どの要求を実行することができるかを決定しなければならない．その次に入出力装置ハンドラは装置に依存する入出力スケジューリングと最適化を行なう．

5.5.3 入出力装置ハンドラ

入出力装置ハンドラは，チャネル・コマンド語の設定，誤り状態の処理，および入出力割込みの処理を行なうほかに，装置の種類によって異なる特殊性に従って精密なスケジューリング・アルゴリズムを実行する．通常，入出力装置の種類に応じて別々の装置ハンドラのアルゴリズムが存在する．

5.5.3.1 回 転 の 順

入出力の負荷が大きいときには，複数の入出力要求が同一の装置を待合せていることがある．前述したように，装置によって，アクセス時間 T_{ij} に大きな偏差の存在することがある．入出力要求の処理の順番によっては，入出力要求を処理する合計時間を減少させることができる．ドラム型の装置に4レコードだけが格納されているとする（図 5.11 参照）．次の4つの入出力要求が受付けられ，この装置への経路が空いているとする：

1. レコード4の読出し
2. レコード3の読出し
3. レコード2の読出し
4. レコード1の読出し

これらの入出力要求の実行は幾とおりかの順序で行なうことができる：

順番A 入出力スケジューラがレコード4, 3, 2, 1の順に読出した場合の処理時間の合計は次のように計算され，約60ミリ秒となる．

$$T_{?4}+T_{43}+T_{32}+T_{21}\approx 3\text{回転}\ (3/4+3\times 3/4)$$

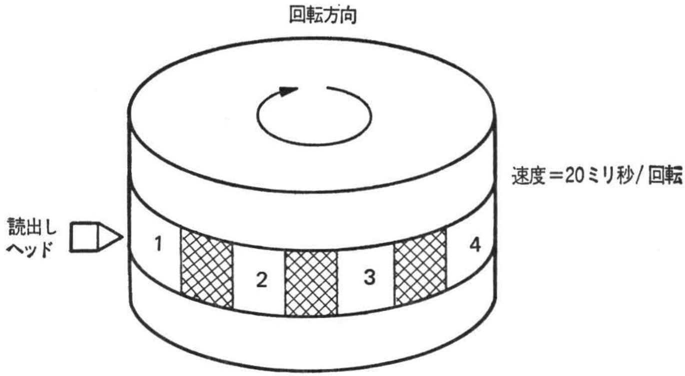


図 5.11 順番にレコードが記録されている単純なドラム状の装置

レコード4の読出しに対しては，ドラムの現在位置がわからないので，1/2回転（10ミリ秒）と仮定し，読出し時間として1/4回転（5ミリ秒）を加える．

順番B 入出力スケジューラがレコード1，2，3，4の順に読出したとすると，合計時間は次のように計算され，約30ミリ秒となる．

$$T_{?1}+T_{12}+T_{23}+T_{34}=1\frac{1}{2}\text{回転}\left(\frac{3}{4}+3\times\frac{1}{4}\right)$$

順番Aと順番Bの結果は同じであるが，速度は2倍となっている．

順番C “？”がレコード3であることがわかったとすると，4，1，2，3，($T_{34}+T_{41}+T_{12}+T_{23}$) の順番がもっとよい．この場合，1回転で20ミリ秒となる．

順番Cによって入出力を行なうためには，装置ハンドラは回転中のドラムの現在位置を知らなければならない．このためのハードウェア機能を**回転位置検出**と呼ぶ．

入出力負荷が大きい場合には，回転順による処理は仕事量を大幅に増大させる（Abate, 1969；Coffman, 1969；Denning, 1969, 1970)．

入出力要求によっては互に排他的なものがあるので，回転の順番で処理するときにいくらか方針を決定しておかなければならない．ドラムに対する要求は，第5.1.2.3.1節で述べたように，トラック番号とレコード番号の2要素から成る番地を指定しなければならない．要求が次のようであったとする：

1. トラック1，レコード1
2. トラック1，レコード3
3. トラック2，レコード2
4. トラック2，レコード3
5. トラック3，レコード1

このとき，要求1と5および要求2と4は，同一のレコード番号を参照するので互に排他的である．したがって，要求1を最初の回転で，要求5を次の回転で，あるいは逆の順で，処理することができる．レコード番号で要求を分類することを**スロッティング**と呼ぶことがある．これは，各回転ごとに各レコード番号の“スロット”が存在することに由来している．同じレコード番号に対する入出力要求は，同一のスロットを“競争”しなければならない．このような競合は，同時に2つ以上のトラックで読み書きできるようなハードウェアを用いれば緩和することができるが，そのためには，普通，チャネルと制御装置および装置の電子回路が余分に必要となる．

IBM 2305 固定ヘッド型ディスクなどの装置では，回転順の処理とスロッティングをハードウェアで自動的に遂行することができる．IBM 2305 の制御装置は，ブロック・マルチプレクサ・チャネルを用いて，最大16個のチャネル・プログラムを同時に受付けることが可能であり，入出力要求を“最適”の順番で処理する．

5.5.3.2 代替え番地

ドラム型の装置で任意のレコードを読出すためのアクセス時間は，回転速度によって大きく左右される．1つの装置ではこの速度は一定である．

各レコードを装置の複数の番地に記録しておくと，実効的なアクセス時間を大幅に減少させることができる．すなわち，同じデータ・レコードを読出すために幾つかの**代替え番地**が存在することになる．この技法は，**フォールディング**と呼ばれる．

1トラック当たり8レコードで回転速度が20ミリ秒の装置を考えよう．レコードAがトラック1のレコード1にあれば，レコードAにアクセスする時間は平均的に半回転の10ミリ秒である．レコードAの写しがトラック1のレコード1とトラック1のレコード5にあり，回転位置の検出を行なって，つねに“最も近い”写しをアクセスすることにすれば，実効的なアクセス時間は5ミリ秒に減少させることができる．同様にして，同一のレコードの写しを多数格納し

ておくことによって，実効的なアクセス時間を2.5ミリ秒あるいは1.25ミリ秒…というように減少することもできる．

もちろん，同一のデータ・レコードの写しを多重に格納すると，実効的な容量は写しの個数で除した値に減少する．レコードごとにn個の写しが存在するとき，この装置はn“重”になっているという．

5.5.3.3 シークの順番

第5.1.2.3.2節で述べたような可動ヘッドの記憶装置では，回転位置のほかにシーク位置の問題が加わる．入出力要求は，シリンダ番号，トラック番号，およびレコード番号の3つの部分から成る番地の指定が必要である．次の要求がそのままの順序で実行されたとしたら，シークで腕を前後に移動させるために多くの時間がかかる：

シリンダ1，トラック2，レコード1
シリンダ40，トラック3，レコード3
シリンダ5，トラック6，レコード5
シリンダ1，トラック5，レコード7

次に示したのはもっと能率のよい順番である：

シリンダ1，トラック2，レコード1
シリンダ1，トラック5，レコード7
シリンダ5，トラック6，レコード5
シリンダ40，トラック3，レコード3

これらの要求は**最小シーク時間の順番**になっている．すなわち，シークとシークの間の距離が最小になっている．

シークの順番にはこのほかにも多くの順番が存在する(Teorey, 1971, 1972)．回転位置の方が重要である場合には，シークの距離は遠くても，回転位置の近いレコードを選ぶ方がよいことがある．たとえば，もしシーク時間が回転時間に比べてごく小さいものであれば，最初の順番は 1，3，5，7 の順にレコードをアクセスしているので最良である．**最小処理時間の順番**を決定することはきわめて複雑となる可能性がある．シーク時間は単調増加関数ではあるが，通常，線型な関数ではないので，順番を決定することは一層複雑となる．腕を20シリンダ分だけ移動させる時間は，10シリンダ移動させる時間の2倍よりは少ないのが普通である．

最小シーク時間および最小処理時間の両方の場合共，順番はこの装置に対する現在の入出力要求によって決定される．最初の要求の実行中に新しい入出力要求が起こることがある．新しい入出力要求のことを考慮すると，元の入出力要求に基づく順番はあまりよくないこともある．システムが先のことまで見とおせる性質のものでない以上，将来起こる要求のことを知るのは不可能である．将来の入出力要求の悪影響を最小にする技法が幾つか採用されている．1つの要求が処理された後に，新しい要求も勘案することができるように，順番を求め直すことが普通行なわれる．**一方向**あるいは**直線的走査の処理**の順番は，一方向に単調に腕を移動させる技法である．最後のシリンダに到着したか，あるいはこの方向にはもう要求が存在しないときには，腕は最初に戻され，再び走査が開始される．この技法は，たえず新しい入出力要求が到着して，腕が前後にはげしく移動するのを減少させようとしたものである．

5.5.4 装置管理のソフトウェアのオーバヘッド

前節で論じた装置管理の技法は装置の効率を向上させるが，処理装置時間の増大を招く（このアルゴリズムがハードウェアで組込まれていない場合には処理時間が増し，ハードウェア化されているときにはハードウェアの経費が増す）．これらのアルゴリズムの価値は，処理装置の速度と入出力要求の量とに関係する．入出力要求が少なければ，処理装置で多重プログラミングを行なうだけで仕事量を大きくすることができる．処理装置の速度が遅い場合には，入出力処理の方で能率がよくなっても処理装置のオーバヘッドの方が多くなる．

たとえば，平均命令実行時間が30マイクロ秒のIBM 360/30で1,000命令の順番変更アルゴリズムを実行すると30ミリ秒かかることになる．順番を変更しないままでの入出力要求の処理が30ミリ秒以下であれば，何ら節約とはならない．しかし，順番の組換え技法は大規模な計算機ではきわめて重要なものとなる．

仮想記憶の管理は，装置管理の負荷を2つの点で増加させる．第1，ページ・スワッピングは大量の入出力要求を引起こすことが多い．第2，たいていのチャネルでは仮想記憶の番地ではなく，物理記憶番地を必要とするので，装置ハンドラは，チャネル・コマンド語を構成するときに，仮想記憶番地を物理記憶番地に変換しなければならない．

最後に，装置管理は，主記憶と2次記憶中の各ジョブの領域の安全を確保しなければならない．極端な場合には，保護のために入出力要求のすべてと参照される番地をきわめて注意深く解析する必要がある．入出力命令は特権命令で

あり，したがって，オペレーティング・システムによってのみ起動が可能であるので，保護が実行できる．IBM 360では記憶領域の保護を行なうために記憶ロックのハードウェアを利用することができる．同様なハードウェアの機能が2次記憶の領域の保護にも用いられることが多い．

要約すると，装置管理のために処理装置の容量のかなりの部分が使用される．しかし，通常は装置管理によって入出力待合せ時間が減少され，システムの仕事量が大幅に増加されるので，装置管理のためのオーバヘッドは支払う価値のあるものである．

5.6 仮想装置

5.6.1 動機

カード読取り機，穿孔機，および印刷機は，効率のよい装置の利用を妨げる2つの困った問題を持っている．

第1，これらの装置はそれぞれ，物理的な特性に適合したような定常的な要求の流れが存在すれば，効率が最良となる（たとえば，1分間にカード 1,000 枚の読込み，1分間に 1,000 行の印刷など）．ジョブが装置の動作速度より速く要求を出すと，ジョブの待合せ時間が大きくなる．一方，ジョブの要求が非常に少ないと，装置は大部分の時間アイドルとなり，有効に利用されない．

第2，これらの装置は一時には単一のジョブに専用させなければならない．たいていのジョブでは何がしかの入力および出力を行なうので，多重プログラミングを行なう分だけのカード読取り機と印刷機が必要となる．このようなことは，たいていのジョブでは装置の能力の一部分しか利用しないので，通常，不経済なことである．

要約すると，これらの装置には定常的な要求を出すべきであるのに，実際のプログラムが出す要求は非常に不規則である．入出力を多量に行なう小さなプログラム（たとえば，Y印を印刷するプログラム）は，数秒の CPU 時間で何千もの印刷要求を発生させる．一方，多量の計算を行なうプログラム（たとえば，素数を1,000個求める）は，CPU 時間にして1時間当たり1行の印刷を行なうだけかもしれない．第2章で述べたように，バッファリングと多重プログラミングはこれらの問題の程度を緩和することはできても，これを完全に解決することはできない．さらに，プロッタ，グラフィック・ディスプレイ，およびタイプライタ端末など，多くの装置でも同様の問題がある．

5.6.2 歴史的な解決法

すべての入力および出力に対して直接アクセス記憶装置を使用することができたとすれば，上述の問題は解決する，あるいは少なくとも大幅に減少する．1台の直接アクセス記憶装置を用いると，図 5.12 に示したように，多数のジョブによる読出し書込みに対して効果的に共用でき，同時使用が可能となる．さらに，直接アクセス記憶装置の速度はきわめて大きいので，特にデータがブロッキングされている場合には，大量の入出力を要求するジョブの待合せ時間を減少させる．

理論的には説得力があるが，すべての入出力に対して直接アクセス記憶装置を使用することは実際的ではないように考えられる．まず最初に入力データを直接アクセス記憶装置に入れるにはどうすればよいであろうか．出力データが直接アクセス記憶装置に記録されているときどうしたらよいであろうか．ディスク・パックを持ってドラグストアに行き，“給与支払の小切手がディスクの特定の部分に入っているからこれを現金化したい”と店員を説得することを考えてみよ．

5.6.2.1 オフライン周辺装置の動作

幸いなことに，このジレンマは図 5.13 に示すように3段階の処理を行なうことによって解決することができる．第1段階では，別の計算機を用いて，カードを全速力で読込み，その情報を直接アクセス記憶装置に格納することだけを受持たせる．転写すべき入力量に応じて，計算機1では2台以上のカード読

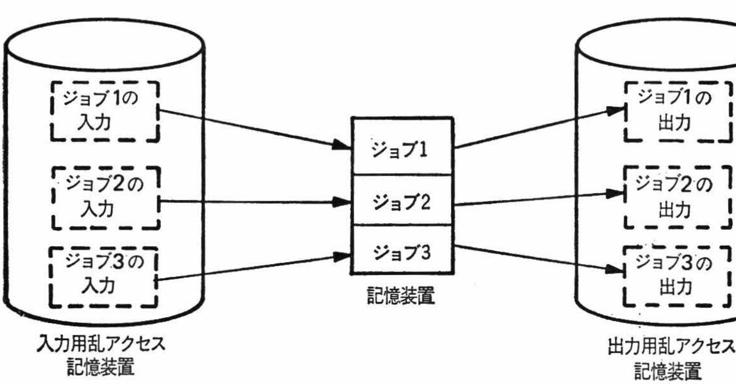


図 5.12 入力および出力に乱アクセス記憶装置を用いる

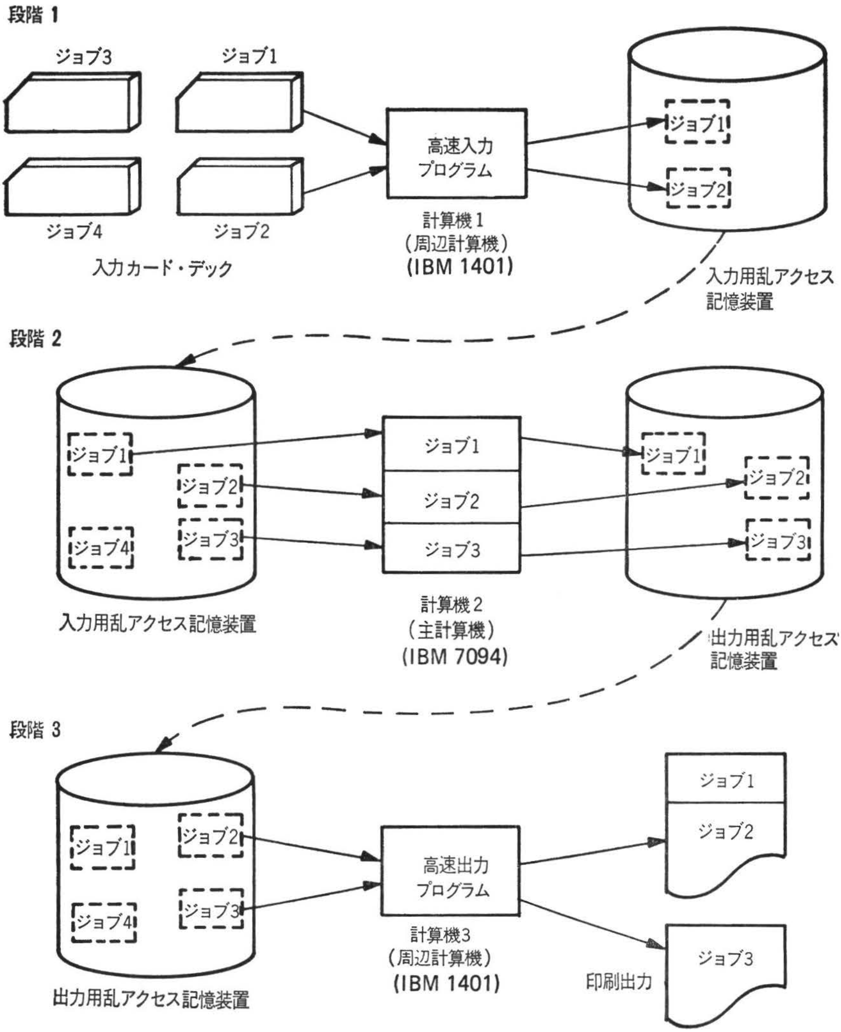


図 5.13 オフライン周辺動作

取り機を使用することができる．

第 2 段階では，計算機 1 で入力が記録された直接アクセス記憶装置は主処理計算機（計算機 2）に移される．この段階は図 5.12 の処理に対応する．この計算機では，複数のジョブが入力用直接アクセス記憶装置から入力を読込み，

出力用の直接アクセス記憶装置に書出しながら，同時に実行できる．この例では，第1段階の計算機1に2台しか物理的なカード読取り機がなくても，3つのジョブを多重プログラムすることができることに注意してほしい．

最後に，第3段階では，出力用の直接アクセス記憶装置は第3の計算機に移され，出力を高速で読出しては印刷機に情報を印刷する．

上述した3段階処理は，1960年代には広く行なわれた．計算機1および3が行なう仕事は**オフライン周辺処理**と呼ばれ，これらの計算機は**周辺計算機**と呼ばれていた．それは，これらの計算機が計算を行なわず，単に情報を1つの装置から別の装置に転送していることに由来する．この動作は主計算機とは独立に実行されるので，“オフライン”動作である．

ここで幾つかの問題を考えることができる．周辺計算機で単純な仕事を行なっているだけであれば，計算機も簡単で，低速で，しかも安価なものでよい．さらに，周辺計算機は必ずしも2台必要ではない．周辺処理の負荷が小さければ，1台の計算機を，数時間おきに入力処理と出力処理とを切換えて，両方の処理を行なうことができる．この反対に，大量の周辺処理が必要ならば，多数の周辺計算機が使用される．

1960年代の始めには，M. I. T. などの計算設備は，1台の主計算機（IBM 7094）と3台の周辺計算機（IBM 1401）とから構成されていた．この当時は，基本的な原理は上で述べたのときわめてよく似ているが，直接アクセス記憶装置の代わりに磁気テープが使用されていた．

オフライン周辺処理技法は前述した問題点を解決することはできたが，次の幾つかの問題が発生した：

1. 人間の介入
2. 返却時間
3. スケジューリング

人間のオペレータが，入力用の直接アクセス記憶装置を入力用の周辺計算機から主計算機へ移し，その後で出力処理させるために必要であり，人間が間違う可能性があったのと能率の悪さに問題がある．主計算機に移す前に，多数のジョブが入力用直接アクセス記憶装置に記録される（ジョブの束）．出力用の直接アクセス記憶装置を出力用の周辺計算機に移す前に，すべての入力ジョブを処理し，出力用の直接アクセス記憶装置を一杯にしなければならない．その後でやっと，個々のジョブの出力を分離して所有者に返却することができる．

このような**バッチ処理**方式は，計算機から見ると能率がよいが，各段階でジ

ョブを行列にして待たせ，返却時間を大きくすることが多い．バッチ処理を行なうと，優先順位によるスケジューリングを行なうことが困難となる．たとえば，2つの優先順位のあるジョブを別々にバッチで実行するとしても，一方のバッチの処理が完了するまでは待合せなければならない．

5.6.2.2 直接結合のシステム

オフライン周辺処理方式の大きな欠点は，入力および出力の直接アクセス記憶装置を主計算機と周辺計算機の間で物理的に移動させなければならなかったことである．この問題を解決するために幾つかの方法が開発されている．図5.14は，入力および出力の直接アクセス記憶装置が，物理的に，周辺計算機と主計算機の両方に接続されていて，人手での処理を不要にしている．この構成を直接結合システムという．通常，主計算機 IBM 7094 の周辺計算機として，IBM 7044 などの1台の強力な計算機が使用される．

直接結合システムの手法を用いると，オフライン周辺処理のたいていの問題

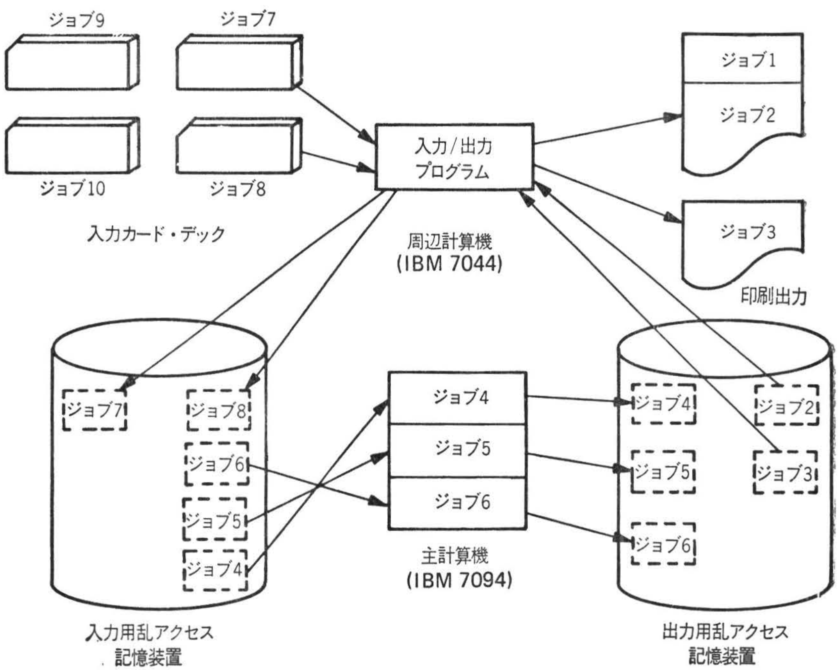


図 5.14 “直接結合システム” の構成

点が解消する．人手の介入は不要であり，周辺計算機が入力用の直接アクセス記憶装置にジョブを入力すれば直ちにジョブ処理を行なうことが可能であるので，“バッチ処理”による返却時間の遅れがなく，スケジューリングの制約もない．

実際の直接結合システムでは，1台の直接アクセス記憶装置を入力および出力の両方に用いたり，あるいは数台の共用の直接アクセス記憶装置を用いることもある．共用される直接アクセス記憶装置を使用するには注意深く整合を取る必要がある．このためには，ジョブが入力用の直接アクセス記憶装置に入れられたこと，およびその格納位置を主計算機に通知する機構が必要であり，出力に関しても同様の機構が必要である．この整合は，システムを大変複雑にする．さらに，周辺計算機と主計算機とが同時に同一の直接アクセス記憶装置を使用しようとしてアクセスの競合が起こり，システムの性能を低下させる．しばしば，直接アクセス記憶装置が重大なあい路となることがある．

5.6.2.3 サポートの処理装置の付加

付加計算機の別の結合法は 図 5.15 **a** に示したように，高速度の結線を用いて主計算機と周辺計算機を結合するものである．この構成法では，周辺計算機はサポートの処理装置と呼ばれる．このサポートの処理装置は，すべての入出力装置や入力および出力用の直接アクセス記憶装置の制御を受持つことになっている．この計算機では，すべてのバッファリングやブロッキングを行ない，主計算機の仕事を単純化している．この場合には，付加された処理装置は，複数の高速カード読取り機や印刷機のように見える（図 5.15 **b** 参照）．前述した仮想記憶の概念に類似して，付加した処理装置は仮想装置を生成する（実際は存在するよりもはるかに多くの装置を利用させる）．

このようなサポートの処理装置を付加する手法の欠点は，2台以上の処理装置が必要となり，処理装置に特定の仕事が割当てられるので（周辺計算機あるいは主計算機），時として1つの処理装置がアイドルとなっても他の処理装置の負荷を処理することが不能であることである．したがって，サポートの処理装置を付加する構成は，必ずしもすべての資源を有効に利用することにはならない．

5.6.2.4 仮想システム

前述の解決法ではすべて，1台以上の特殊計算機が入出力機能の処理のために専用されていた．入出力チャネルの強力な処理能力とシステムの多重プログ

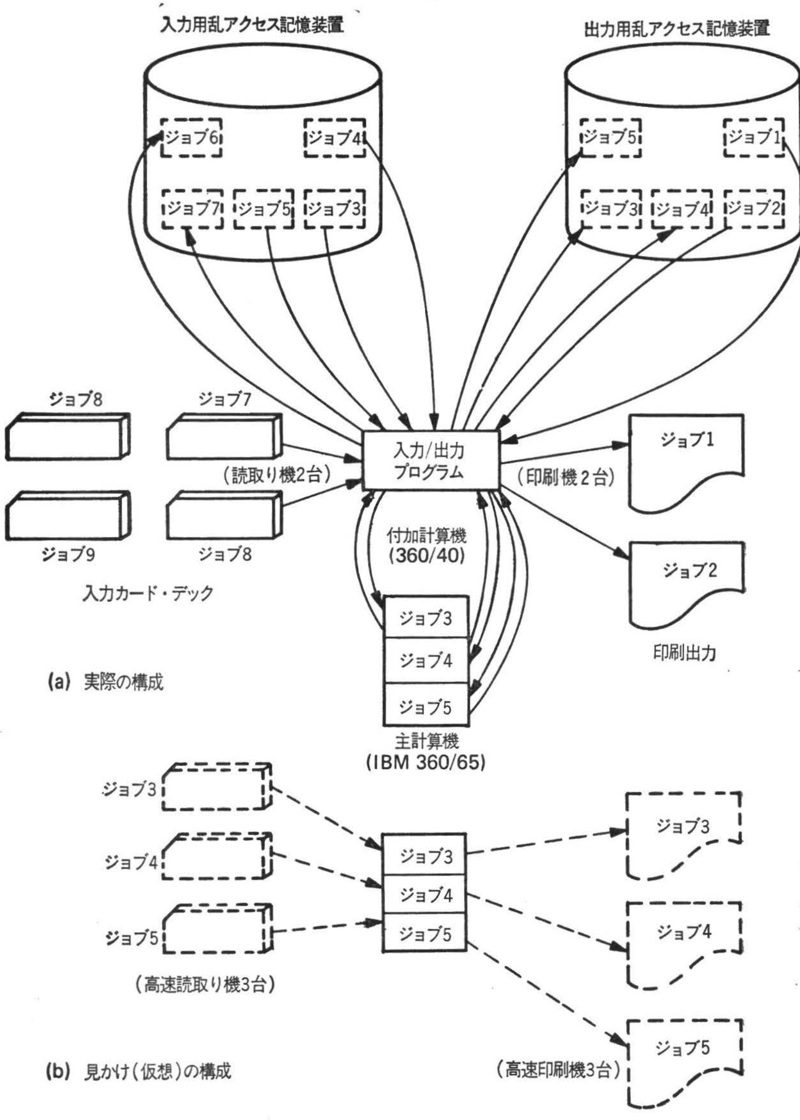


図 5.15 "付加計算機" の構成

ラミング能力とを考慮すると，入出力の処理のために別個の計算機システムが本当に必要なのであろうか．答えは否である．スプーリングを行なうシステムでは，図 5.16 に示したように，主計算機がオンライン同時周辺動作(SPOOL)を実行する．

このようなシステムでは特定のジョブがサポート計算機で達成された機能を遂行するために割当てられ，主計算機中に常駐して，多重プログラミングの下で通常のジョブと主計算機を共用する．これらのジョブは実は利用者ジョブでなくシステム・ジョブであるから，“幽霊”とか“悪魔”とかいう特別の名前がつけられている．近代的なオペレーティング・システムでは，多くの特殊機能はこのような特別のシステム・ジョブによって実行れる．

本節では主として，スプール・システムを装置管理および仮想装置の観点から調べて行くが，このシステムには，広く記憶管理，処理装置管理，装置管理

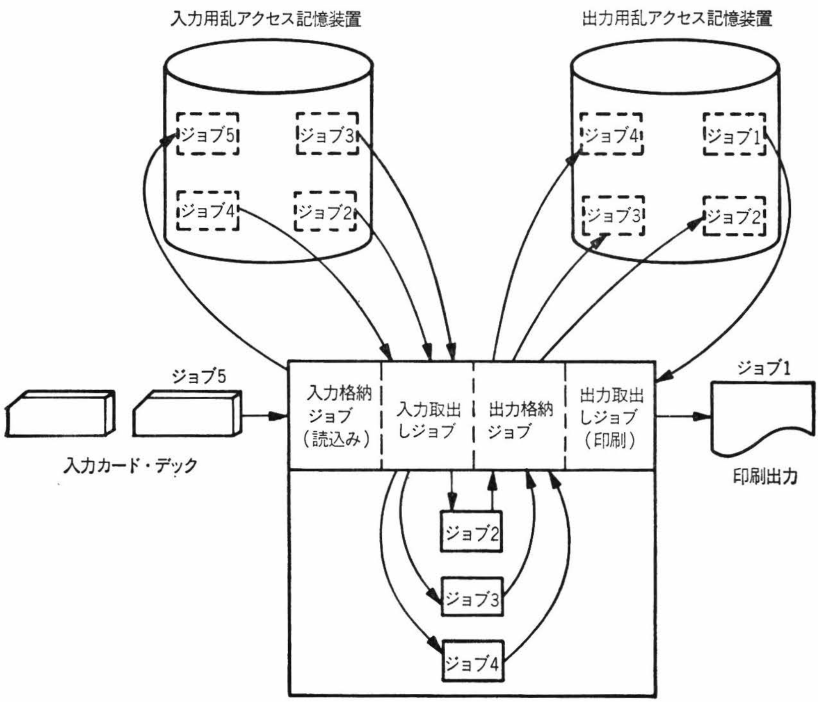


図 5.16 スプーリング・システム

および情報管理と関係する複雑な機構が含まれることに留意してほしい．スプール機能を果たすジョブは特別の恒久的なシステム・ジョブであるから，記憶管理はこれらのジョブを利用者ジョブと異なった方法で処理する．実際，システムによっては，これらのジョブは，オペレーティング・システムの記憶領域の中に編入されている．これらのジョブの間では通信と調整が必要であるから，処理装置管理の各種の機能を使用しなければならない．直接アクセス記憶装置に格納されている入力および出力のデータは情報管理によって使用されているのと類似の機構を用いて管理しなければならない．最後に，性能を高めるために，効率のよいバッファリング，ブロッキング，および入出力制御を行なわなければならない．

スプール機構の組込みと利用できる手法の難易は，オペレーティング・システムの記憶管理，処理装置管理，情報管理，および装置管理が提供する機能に直接依存する．総合的なオペレーティング・システムにおいては，スプール・プログラムの役割りはむしろ小さなものであり，オペレーティング・システムの残りの部分がもっと“大変な仕事”を行なっている．一方，もっと限定されたオペレーティング・システムでは，スプール・プログラムはオペレーティング・システムの多くの機能を実行しなければならない．実際，スプーリングはシステムの性能をしばしば大幅に改善するので，利用者ジョブの多重プログラミングや汎用の情報管理を行なわない単純なオペレーティング・システムの一部として装備されることが多い．このような場合には，スプール・プログラムは自分自身で多くの処理を行なわなければならない．

5.6.3 スプーリング・システムの設計

本節では，簡単なスプーリング・システムの設計を行なってみよう．スプール・プログラムは，（“通常の”ジョブではなく）オペレーティング・システムの1つの構成要素であり，独自の情報管理を自分で行なうものである．このような考え方は，小規模から中規模，場合によってはIBM VM/370システム（第9章参照）のような大規模のオペレーティング・システムで行なわれている．これらのシステムでは，次の2つの特別のオペレーティング・システムの機能（スーパバイザ呼出し）が用意されている：

1. 入力カードの読込み
   CALL READNEXT (BUFFER)
2. 出力行の印刷

CALL PRINTNEXT (BUFFER)

もちろん実際には，仮想装置が使用されているので，これらの呼出し時にはカードは物理的に読込まれないし，行の印刷も行なわれない．この区別は，これらの要求を発するジョブからはわからないようにするべきである．

一般的な入力および出力のスプール・システムは，図5.17に示したように，4つの構成要素に分割することができる．これらの個々の要素は，機能（入力あるいは出力）あるいは制御の移され方（呼出しあるいは割込み）などによって幾とおりかに区分することができる．プログラムを“呼出しによるもの”と“割込みによるもの”とに分割することは，各種の装置や，オペレーティング・システムの機能を取扱ったりする場合に一般的に行なわれることである．以下の議論はスプール機能を強調しているが，読者は手法の一般性についても留意してほしい．

5.6.3.1 入力スプール

設計を簡単にするために本書では入力スプールの機能だけを解析する．出力

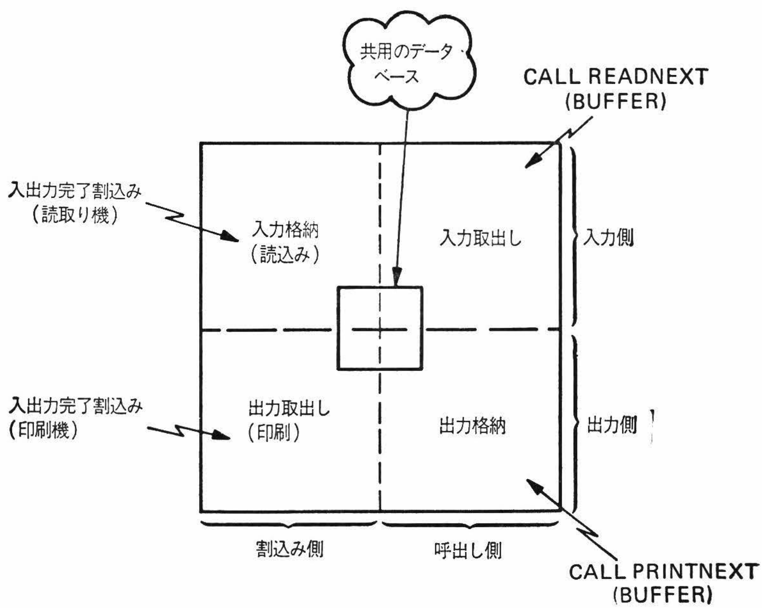


図 5.17 スプーリング・システムの一般的な構造

スプールも同様に取扱うことができる.

入力スプール機能によって実行しなければならない2つの動作は次のとおりである：

1. 各入力カードを物理的に読込み，直接アクセス記憶装置に格納すること
2. 実行の間ジョブが，直接アクセス記憶装置の中の入力カードの写しをアクセスできるようにすること

後者の動作は，実行しているジョブが READNEXT の呼出しを行なうことによって遂行される．しかし，読込みと格納動作はいつ遂行されるのであろうか．この動作はジョブが開始する以前に完了していなければならないので，ジョブとは独立に行なわれる．カード読取り機から物理的に読込んだ入力カードの格納は，入力が完了すると即座に行なわれる．したがって，この動作は，カード読取り機から入出力完了の通知が行なわれるのに応じて起動される．この種の動作は**割込み駆動型**と呼ばれる．第2章で示した入出力バッファリングのプログラムは，割込み駆動の機構の別の例である．

入力スプールの呼出しによる処理機能と割込みによる処理機能は互に関連して働く．この関係と調整は，図 5.17 に示したように，共用のデータベースを用いて処理される．このためのデータベースは図 5.18 に示されている．

直接アクセス記憶装置は，2,000 枚程度の適当な容量の入力デックを収容することができる部分に分割される．1台の IBM 3330 型のディスク装置を用いて，1つのシリンダに1つの入力デックを収容することにすると，最大 400 の入力デックを収容できる（各デックはカード約 2,000 枚である)．第6章ではもっと複雑な格納技法を示す.

図 5.18 の主要なデータベースは入力スプール表である．この表は，入力用の直接アクセス記憶装置の各記憶領域（通常，スプール領域と呼ばれる）の状態を示している．1つのスプール領域が未使用のときは，**空き**状態である．入力デックを収容するために使用中の場合には，**入力**，**保持**，あるいは**実行**状態である.

入力　入力デックは読込み中である.

保持　入力デックは完全に直接アクセス記憶装置に複写されたが，対応したジョブはまだ実行が開始されていない.

実行　対応したジョブは現在実行中で，入力データをスプール領域から読出している.

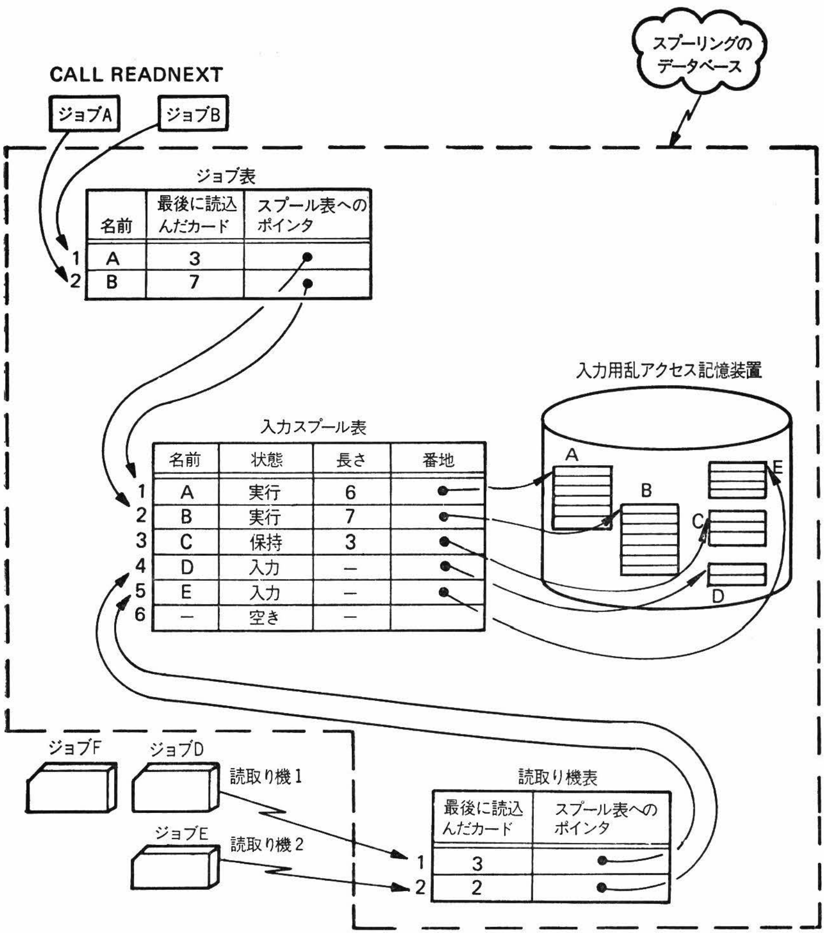


図 5.18 スプーリング・システムの主要なデータベース

上述の状態情報のほかに，スプール表には，入力デックの長さと，記録および課金のために利用者がつけたジョブ名も格納される．最後に，スプール領域の場所も記される．本節の例では，この場所は，入力デックの写しを収容しているシリンダの番地で示されている．

5.6.3.2 スプーリングとジョブ・スケジューリングの関係

ちょっとここで脱線をすることにしよう．処理装置管理のジョブ・スケジューリングの議論では，オペレーティング・システムは実行すべきジョブを知っており，次に実行するジョブを任意に選択できると仮定していた．ジョブ・スケジューラが，計算機のオペレータに問合せたり入力デックを手でかき回したりせずに，このスケジューリングを行なうにはどのようにすればよいか．

スプール機能はこれらの問題点の大部分を解決する．“保持”状態になっている入力スプール領域は，実行しようとするジョブに対応している．もう少しうまくやると，入力スプールのプログラムは，ジョブ・カードに記入された CPU 時間の予想値，優先順位，その他のジョブのパラメータを入力スプール表の対応した欄に設定することができる．この場合には，スプール表はジョブ・スケジューラのための重要なデータベースとしても使用でき，**ジョブ行列**とか**ジョブ保持表**とか呼ばれる．

さらに，スプール機能によって実現される仮想のカード読取り機を利用すると，入力デックを物理的にかき回さずに，任意のジョブの実行を開始したり，入力データを読込んだりすることができる．したがって，ジョブ・スケジューラとスプール機能とは，全く異なった目的のために作られたものではあるが，重要な関係がある．実際，システムによってはこれらの問題を一緒にして処理している．

5.6.3.3 入力スプールのアルゴリズム

入力スプール表に加えて，次の 2 つの補助的なデータベースがある．

1. **読取り機表**　物理的なカード読取り機とそれに対応して用いられているスプール領域の状態情報を格納する．
2. **ジョブ表**　現在実行しているすべてのジョブとそれに対応する仮想装置として使用されているスプール領域の状態情報を格納する．

データベースはスプール・システムの設計の最も重要な部分である．実際のルーチンは，これらの表を修飾したり表を使用したりするような，むしろ簡単明解な動作を実行する．全体のアルゴリズムは 図 5.19 (a および b) に示されている．呼出しあるいは割込みによってシステムに制御が移されると，それぞれジョブ表あるいは読取り機表の欄が決定される．次に，スプール表の欄とスプール領域が決定される．いずれの場合にも，スプール領域のブロックは順

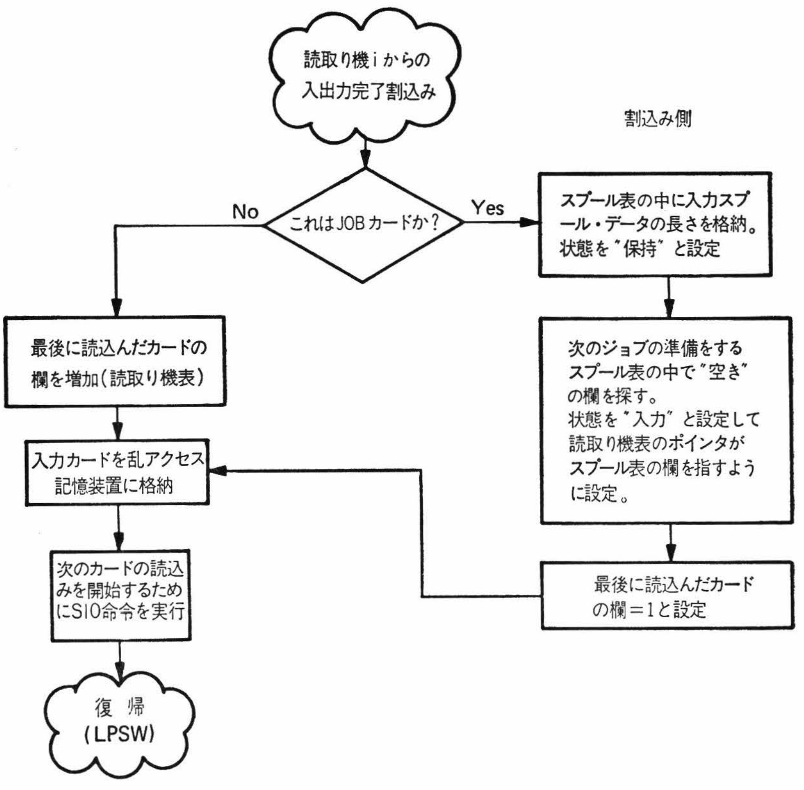


図 5.19 a　入力スプール・プログラムの割込み側の処理

番に処理される．たとえば，直接アクセス記憶装置が，1トラック当たり80バイトのレコード100個，1シリンダ当たり20トラックに編成され，1つのジョブに1つのシリンダが割当てられるとすれば，スプール表にはそのシリンダ番号が記入され，トラック数，およびレコード数は次のようになる：

$$\text{トラック番号} = \frac{\text{カード番号}}{100}$$

$$\text{レコード番号} = \text{剰余}\left[\frac{\text{カード番号}}{100}\right]$$

したがって，カード 1=(トラック 0，レコード 1)，カード 105=(トラック

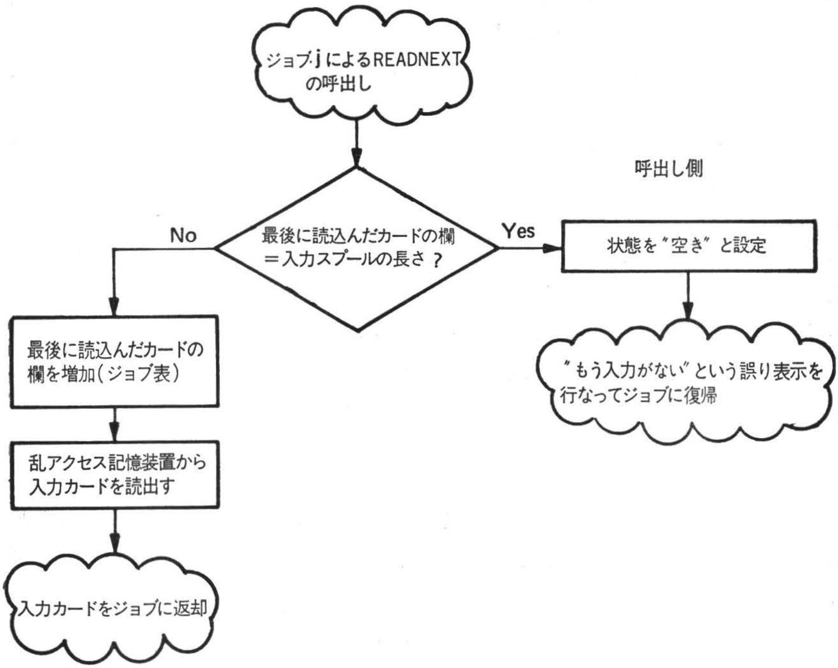


図 5.19 b　入力スプール・プログラムの呼出し側の処理

1, レコード 5), …となる.

図 5.19 a は入力スプール・プログラムの割込みによる処理機能で使用されるアルゴリズムを示している. 読取り機から入出力完了割込みが起こると, 適当なスプール領域の中の次のレコードの場所に入力カードが格納される. 割込み発生点に制御を復帰する前に, 次の入力カードを読込むための入出力が起動される. スプール・システムで 1 分間に 1,000 枚の読取り機を 1 台使用していたとすると, ほぼ 60 ミリ秒に 1 回, 割込み処理の入口が呼出されることになる.

割込み処理機能では, 各入力デックの開始を示し入力デックを区別するために用いられる特別のジョブ・カードの識別をも行なう. このようなカードが読込まれたときには, 前の入力スプール領域は完結したと考えられ, "保持" 状態にされる. 次の新しい入力デックのためにスプール表の "空き" の欄とスプール領域を求めなければならない. 次に, この欄は "入力" 状態と設定され,

ジョブ名などの情報がジョブ・カードからスプール表の欄に転送される．読取り機表の欄が次に更新されて正しいスプール領域を指すように設定され，“最後に読込んだカード”の欄は1に設定される．これらの手順が完了すると，正規の割込み側の処理が上述のように実行される．

呼出し側は，利用者ジョブが READNEXT を呼出したときの要求に基づいて動作する．ジョブ表の中でこのジョブの欄を求め，すべてのカードが読込まれてしまっていないかどうかを検査する（最後に読込んだカード＝入力の長さ）．すべてのカードを読込んでしまった場合には，スプール表の欄は“空き”状態に設定され，ジョブには特別の誤り返却符号が返される（PL/I プログラムでは ENDFILE 条件と呼ばれている）．それ以外の場合には，直接アクセス記憶装置のスプール領域から次の入力カードが読込まれてジョブに返却される．

5.6.3.4 入力スプール・アルゴリズムの特別な場合

上述の設計は，必ずしも効率がよくはないが，スプール・システムの方式として利用できるものである．システムには考慮しておかなければならない特別の場合が幾つか存在する：

1. 割込み処理機能を開始させるにはどのようにすればよいか．
2. 読取り機が誤動作したり読込むカードが存在しないために一時的に停止したら何が起こるか．
3. これ以上スプール表の欄やスプール領域の空きがなく，次の入力デックを保持できないときには何が起こるか（入力の速度に比べてジョブの実行が遅いと“保持”状態のジョブ入力が数百数千になることがある）．
4. “実行”あるいは“保持”状態のスプール表の欄が存在しないときには何が起こるか．

1. および2. の場合は同一の基本原理で処理することができる．最初に入出力の起動を行なう必要がある．入出力完了割込みが発生したとき，割込み処理機能は，“仕事に戻る”．この最初の入出力起動は2つの方法で行なうことができる：

1. START READER のような特別なオペレータ・コマンドに応答して行なう．
2. **入出力実行可割込み**に応答して自動的に行なう．この割込みは，IBM 2501型のカード読取り機などで読取り機のホッパーにカードが置かれて

起動ボタンが押されたとき発生する.

3および4の場合は，呼出し処理機能と割込み処理機能との協同作業が必要である．スプール表に空きがないときには，割込み処理機能は一時的に読込みを停止して，入出力起動を行なわず単にそのままで割込まれたプログラムに制御を復帰しなければならない．スプール領域に対して使用される直接アクセス記憶装置の容量が十分に大きいときにはこのようなことはまれである．スプール表の欄が空いたとき呼出し処理機能は割込み処理機能に通知しなければならないので，協同作業が必要となる．このとき，割込み処理機能は，空いたスプール領域にジョブ・カードを格納し，スプール表の欄の状態を“入力”とし，次の入力カードを読込むために入出力起動を行なうことができる．

4の場合は3の場合と同様にして処理される．“実行”あるいは“保持”状態のジョブが存在しないときには，処理装置が行なう仕事がない．したがって，処理装置は待合せ状態となるが，割込みは可能となっている．入力が読込まれているときの入出力完了割込みや，読取り機がアイドルであった場合の入出力実行可割込みが発生すると，処理装置は実行を開始し，スプールの割込み処理機能に制御を移す．割込み処理が終了すると，その処理の終りに実行されるプログラム状態語ロード命令（LPSW）によって再び待合せ状態となる．

入力デックが完全に読込まれ，“保持”状態とされると，呼出し処理機能，すなわちもっと正確にいうとジョブ・スケジューラが目覚される．これは，割込み処理から LPSW 命令を用いて制御を復帰する前に待合せ状態のビットをオフにすることによって行なわれる．

図 5.20（aおよびb）は，上で論じた場合を処理するように変更した呼出しおよび割込み処理機能のアルゴリズムを示している．オペレーティング・システムの他の要素，特に処理装置管理，の機能によっては，別の組込みが適していることもあるという点をもう一度注意しておかねばならない．たとえば，第7章の例題のオペレーティング・システムでは，呼出しおよび割込み処理機能の間の調整は，セマフォアとPおよびVオペレータによって行なわれている．

5.6.4 スプール・システムの効率の考察

これまでのところでは，スプール・ルーチンによって実行される直接アクセス記憶装置の入出力は，バッファリングも，ブロッキングも，同時実行も行なっていないと仮定していた．物理的な入出力レコードに対して1つの80バイトの論理レコードが対応しているものとしていた．直接アクセス記憶装置のレコ

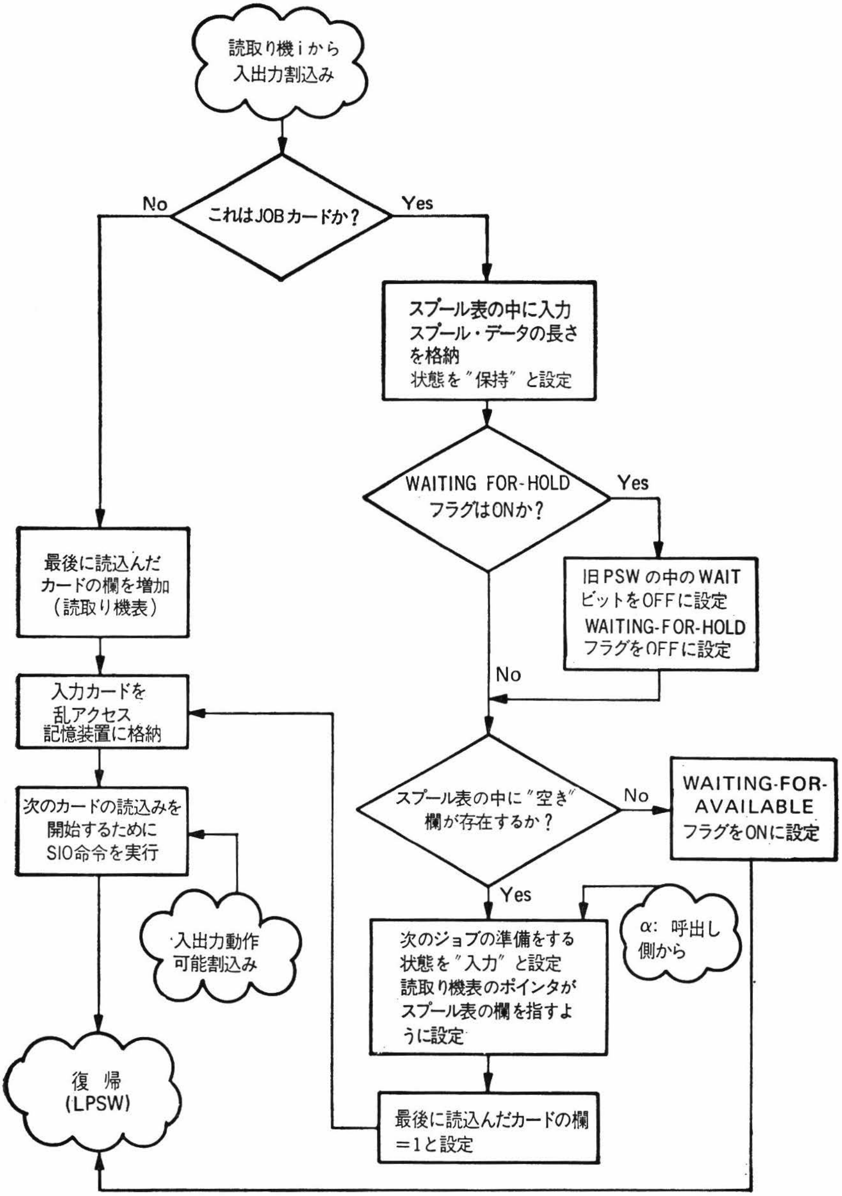


図 5.20a 入力スプール・プログラムの割込み側の処理（改訂版）

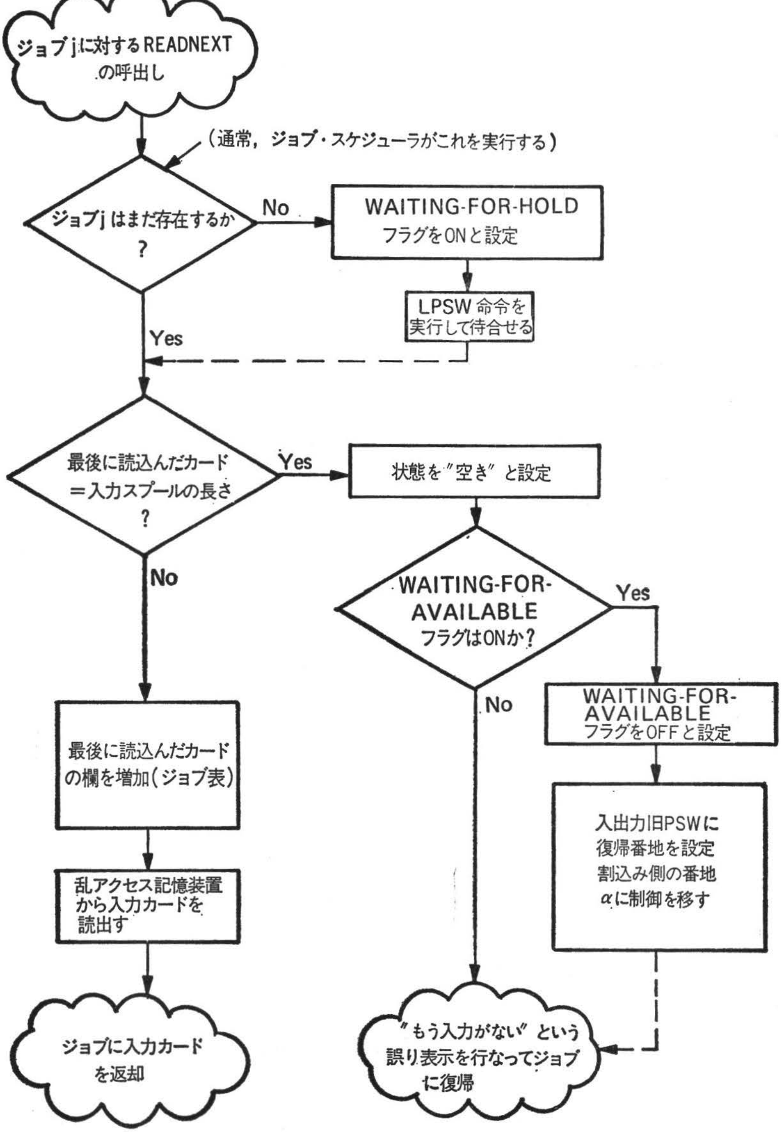


図 5.20 b　入力スプール・プログラムの呼出し側の処理（改訂版）

ードを読み書きする必要があるとき，READNEXT および PRINTNEXT の場合は呼出し者のバッファを使用し，割込み処理機能の場合には読取り機や印刷機に関連した1つの領域が使用された．さらに，スプール・プログラムが直接アクセス記憶装置の入出力を待合せている間，システム全体が停止していた．本節ではこの方法の効率を考え，別の方法を考察する．

1分間1,000枚の速度では，カード読取り機で60ミリ秒以内に次のカードを読込むことができる．直接アクセス記憶装置はどれだけ速いであろうか．先にも述べたように，IBM 2314 や IBM 3330 型などの大容量の直接アクセス記憶装置で任意の80バイトのレコードを読出すには，ほぼ60ミリ秒かかる．したがって，複数の仮想装置を生成するために直接アクセス記憶装置を使用することができるが，特に速くなるというわけではなさそうであり，実際遅いこともある．

直接アクセス記憶装置で80バイトのレコードを読出すのに60ミリ秒かかるが，80バイトの論理レコードを10個含む1つの800バイトのレコードを読出すのに約61ミリ秒しかかからないことに気がつかなければならない．スプーリングで直接アクセス記憶装置を読み書きする場合にはレコードは必ずスプール領域の中で順番になっており，使用中のスプール領域（入力，出力，実行）のためにそれぞれ800バイトのバッファを割当て，800バイトのブロックを読み書きし，バッファを80バイトのレコードにデブロックすれば，実効的には6ミリ秒のアクセス時間を得ることができる．これは物理的なカード読取り機の10倍の速度である．すなわち，スプール領域に対する最初の READNEXT 要求は60ミリ秒かかり，第2番から第9番の要求は，デブロッキングとその他のスプール・プログラムの計算時間を加えても1ミリ秒以下しかかからない．第10番の要求は，再び60ミリ秒かかる．したがって，平均的には，60ミリ秒ごとに10個の要求が満たされる．8,000バイトのレコードを使用することにすると，実効アクセス時間は1ミリ秒に低下する．これは物理的なカード読取り機の60倍の速度である．スプール要求を処理するためのソフトウェアのオーバヘッドは1ミリ秒に近く，いくらブロッキングをしても，この値を減少させることにはならないので，8,000バイト以上のブロックを用いてもこれ以上速度の向上は期待できない．

本章の始めに述べたように，大きな物理レコードを用いると，レコード間隙の個数を減少させることとなり，直接アクセス記憶装置の領域をより効率よく利用することができる．したがって，大きなレコードを用いると，スプール領域の個数や容量は大幅に増大する．

入出力時間を他のジョブの実行と重ねるように多重プログラミングを行なうことができる．すべてのジョブが同時に入出力を待合せている場合には，システムは一時的にアイドルとなる．バッファを2つ使用してダブル・バッファリングを行なうと，1つのブロックをデブロッキングしている間に次のブロックを読出して，入出力動作による遅延を一層減少させることが可能となる．

上述の技法を用いると，仮想装置の実効アクセス時間を“実”の装置の時間の数分の1にすることができる．

スプール技法はこのようにすばらしい特性を与えるのに，どうしてこれがすべてのシステムで利用されないのであろうか．これには多くの理由がある．1つには，効果的なスプール・プログラムは複雑で，オペレーティング・システムの領域をかなり必要とする．また，かなりの容量の直接アクセス記憶装置の領域がスプール領域として必要である．これらに関する見当をつけるために，スプールで使用されるバッファの主記憶容量を考えてみよう．システムには，2台の読取り機および3台の印刷機があり，6つのジョブが実行されているとする．また，ダブル・バッファリングが行なわれ，ブロックの大きさは 8,000 バイトであるとする．バッファのために主記憶容量がどれだけ必要となるであろうか．ダブル・バッファリングを行なうと，各読取り機および印刷機に2つのバッファが，また，実行中の各ジョブには4つのバッファ（入力に2つと出力に2つ）が使用される：

読取り機	2台	2×2=4	バッファ
印刷機	3台	3×2=6	バッファ
ジョブ	6	6×4=24	バッファ
		34	バッファ
		×8,000	バイト／バッファ
		272,000	バイト

もちろん，小さなブロックを用いて，単一バッファリングを行なうと，この容量を大幅に減少させることができる（800 バイトのブロックで単一バッファリングを行なうと，わずか 13,600 バイトですむ）．しかし，スプール・システムのために主記憶容量が多く必要である．1,000,000 バイトのシステムは大規模システムであると考えられており，最大記憶容量が 272,000 バイト以下の中小規模の計算機が多数存在することを考えると，270,000 バイトという数は劇的な数である．したがって，スプーリングの価値は，与えられたシステムにおける有用性と経費との比較において決定されなければならない．

5.7 将来の装置管理

装置をもっと効率よく利用するためにソフトウェアで利用できる多数のハードウェアの機能の出現が期待される．たとえば，IBM 370では，その前の360にはなかったような，回転位置検出やブロックの多重化を実現した．装置によっては，自動的な回転順の並べかえが可能である．将来は装置管理のソフトウェアの機能の多くがハードウェアによって組込まれ，装置の仮想化の技法がもっと広く行なわれるものと予想される．

5.8 要　　約

本章では，装置管理の4つの基本機能を処理する技法について述べた．これらの機能を処理するための枠組を，次の3つのソフトウェアのモジュールを基に，述べた．

1. 入出力トラフィック制御
2. 入出力スケジューラ
3. 入出力装置ハンドラ

読者は，これが唯一の，あるいは，最善の方法であると考えてはならない．しかし，この枠組みは原理的には最も適当と考えられ，今日の多くのシステムで行なわれている．第7章の例題のオペレーティング・システムは，装置管理ルーチンの明解な構成を示している．

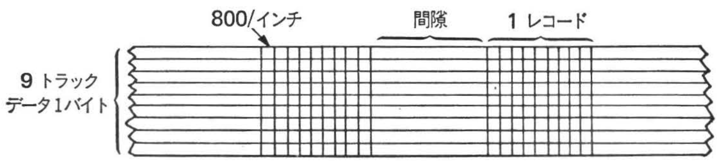
演 習 問 題

5.1 磁気テープの各物理レコードの間には**レコード間隙**（IRG）が存在することを知った．この IRG は，主として磁気テープ装置が起動してデータを読み始める前に一定の速度に達し，また，ここで停止することができるようにするためである．一方，ディスクやドラムはつねに回転しているので，起動や停止のための領域を必要としない．

1. ディスクやドラムではハードウェア的に IRG が必要か，あるいは，あった方が都合のよいことがあるか．
2. ディスクやドラムの IRG は，ソフトウェアとして役に立つか．

5.2 可変長のレコードを持つ磁気テープ記憶装置を考える．レコードの開始は，レコード開始マーク，レコードの終了はレコード終了マークにより指示されるものとする．

磁気テープ装置は200インチ／秒の速度でヘッドの下にテープを移動させるものとする．入出力動作を入出力動作の間でテープは停止しなければならない（1レコードの読出しまたは書込み）．駆動装置は，40,000インチ/秒^{2}の割合でテープを加速したり減速したりする．情報の書込まれている密度は800ビット／インチであり，情報の書式はデータ8トラックと1つのパリティ・トラックとである．



1. この装置の情報転送速度はどれだけか．すなわち，テープが一定の速度となってからは，幾らの割合（バイト／秒）で情報を読み書きすることができるか．

2. レコードを順番に読むとして，“局所的”なアクセス時間（一定の速度となる時間）は幾らか（秒）．

3. IRG の長さは幾らか．

以下の問では，IRG の長さを3/4インチと仮定せよ．

4. 1レコード当たり80バイトのカード・イメージを書込むとすれば，情報が記憶されるテープの割合は何%か，1レコード当たりカード100枚のときはどうか．

5. ブロッキング係数とテープの使用率との関係をプロットせよ．ただし，1レコードにカード10枚分を記憶するとき，係数を10とする．

6. データ領域として残っている主記憶領域が24,000バイトであるとするとき，テープ上の情報の割合の最大値は幾らか．この値からして，磁気テープは経費的に効率が悪いか．

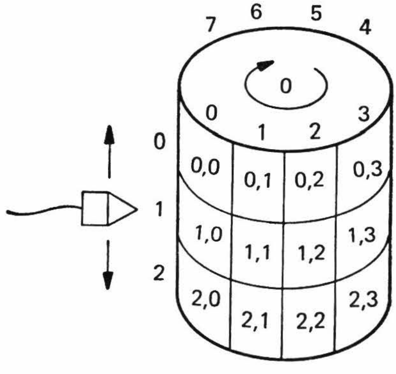
7. 長いレコードにデータをブロッキングすることの3つの問題点は何か．

5.3 可動ヘッドの磁気ドラムがあるとする．ヘッドが読み書きできるのは3

つの位置(トラック0, 1, 2)である．各トラックには8つのレコード(0〜7)が記憶される．ドラムは8ミリ秒で1回転する．ヘッドも8ミリ秒で隣接のトラックに移動する．

ドラムを取扱うルーチンには，次のリストに示す番地に対する読込み要求が受付けられている：

トラック, レコード
0,2
1,2
1,3
1,5
2,4
2,3



1. 上の入出力要求をそのままの順番 (FIFO 方式) で処理するとどれだけの時間がかかるか．この問では，読出しを始めるときにドラムとヘッドの位置が(0, 0)にあると仮定してよい．
2. これらの要求を処理する最適な順番は何か．使用したアルゴリズムを言葉で説明せよ．
3. ドラムに対する入出力要求を最適化するルーチンの中で，次に示したものがいかに使用されるかを説明せよ．
   a ドラムとヘッドの位置を検出する命令．
   b 指定されたトラックにヘッドを移動する命令．この命令が実行されたら直ちにチャネルを解放する (動作の完了を待合せる必要はない)．

c　ヘッドの移動時間が2ミリ秒のドラム.

4. 次に示したことは，ドラム・システムの応答速度にどのような影響を与えるか.

a　より多くのトラックを追加すること.

b　ドラムの反対側に第2の可動ヘッドを追加すること.

5.4　次の表は4種類の記憶装置を示している．各装置は，順アクセス，完全な直接アクセス，あるいは直接アクセスのいずれに当たるかを第1欄に示せ．第2および第3欄に示した特性で比較してみよ．第4，第5，および第6欄に答えよ．専用，共用，仮想のいずれの技法が最適かについて論じよ.

	1	2	3	4	5	6
記憶装置	順アクセス，乱アクセス，あるいは完全な乱アクセス	記憶容量	平均アクセス時間(乱アクセス)	レコードの番地指定法	レコードの位置決め法	装置に適当な応用例
磁気テープ						
可動ヘッドのディスク						
固定ヘッドのドラム						
拡張コア記憶装置						

5.5

1. 入出力トラフィック制御の機能は何か.
2. 入出力装置ハンドラの機能は何か.
3. 入出力スケジューラの機能は何か.
4. トラフィック制御はどのプロセスの番地空間に入っているか.
5. 装置ハンドラはどのプロセスの番地空間に入っているか.

5.6 システム設計者の1つの目標は，効率を上げること，すなわち仕事量を増大させることである．本章では，システム性能に対する，種々のジョブおよびプロセス・スケジューリングの効果を論じた．ジョブの入出力のすべては，入出力スケジューラでスケジュールされるので，入出力スケジューラもシステムの性能に大きな関係がある．

1. 入出力スケジューラの性能を改善すると，システム全体の性能に大きな関係があるような条件を述べよ.
2. 入出力スケジューラの性能を改善しても，システム全体の性能にほとんど関係がないような条件を述べよ.

5.7 オペレーティング・システムにおいて，装置の専用割当ての技法を採用せねばならないような問題点を2つあげよ.

5.8

1. 1つの装置を共用するときの問題点を3つ示せ.
2. ディスクが磁気テープに比して共用がやさしい理由は何か.

5.9 テレタイプなどの通信用の装置も，原理的には，他の入出力装置と同様に取扱うことができる．しかし，2，3の点で注意しなければならない特別の特性が存在する．第1，出力の速度がかなり遅く，毎秒10字である．80バイトの行を印刷するには8秒かかる．第2，入力の速度は人間のタイピストの速度に制限される（タイピストは行の途中で昼食のために休憩することさえある).

通信用の装置に対しては，主として3種のハードウェアのインタフェースが存在する：

1. 割込み
2. ポーリング
3. バッファリング

割込み型に対しては，鍵盤が押されたとき割込みが発生する．このとき

割込みハンドラは入出力起動命令を実行してデータ・バイトを読込まなければならない．**ポーリング型**に対しては，入出力起動命令を実行すると，鍵盤が押されていたときにはデータ・バイトが返却され，鍵盤が押されていなかったときには誤り状態が通知される．**バッファリング型**に対しては，入出力起動命令が実行されたとき，文字がタイプされるごとにチャネルが自動的に，メッセージ終了キー（通常は改行復帰キー）が押されるまで，記憶番地へ順番に文字を転送する．

これらのハードウェアのインターフェースのそれぞれについて，主要な問題点を示し，対応する装置ハンドラの設計を示せ．

5.10 遠隔端末の入力／出力を処理するには 文字単位あるいは行単位の2つの方法がある．文字単位のモードでは，各文字ごとに別個の入出力起動命令が実行され，その都度割込みが発生する．行単位のモードでは，各行ごとに入出力起動命令の実行と割込みの発生がある．一般に，行モードの方が，割込み処理が少なく，時間関係の条件も少ないので，効率がよい（文字モードでは，1つの割込みの処理と次の入出力起動命令の実行とは，割込みが発生してから100ミリ秒以内に完了しなければならない）．

1行は最大132文字と仮定する．次の問に答えて，役に立つと思うこととその利用法について述べよ．

1. 小規模なタイムシェアリング・システムに接続されている5台の端末装置を取扱うのに最もよいモードはどれか．またその理由は何か．
2. 航空機の座席予約システムに接続されている4,000台の端末装置を取扱うのに最もよいモードはどれか，またその理由は何か．
3. 上の方式を折衷することは利益があるか（たとえば，16文字入力されるか，あるいは行が終了すると割込みが起こる）．理由を説明せよ．

5.11

1. バッファリングの技法とブロッキングの技法との相違点は何か．
2. これらの2つの技法の応用例を示せ．
3. 各技法の利点を3つあげよ．

5.12 スプーリングを行なうシステムの多くでは，入力デックが**すべて**直接アクセス記憶装置に複写されないとジョブを開始することができない．また，ジョブが終了するまでは，出力を開始することができない（すなわち，出力が**すべて**直接アクセス記憶装置に複写されるまでは出力されな

い).

本問の問1，2および3では次に示す5つのジョブ（各100Kバイト）を考える：

ジョブ番号	実行時間	入力カード数	出力行数
1	3 分	100	2,000
2	2 分	200	600
3	4 分	1,000	4,000
4	1 分	2,000	400
5	6 分	700	5,000

これらのジョブは表に示した以外の入出力を行なわず，ジョブの実行中を通じてほぼ均一に入力カードを読込み，かつ出力行を印刷するものと仮定する（すなわち，ジョブは，必ずしも最初に入力をすべて読込み，終りに出力をまとめて書出すことをしない).

さらに，これらのジョブは，カード読取り機1台と印刷機を2台備えている多重プログラミングの計算機システムで実行される．システムに1Mバイトの主記憶があるので，5つのジョブはすべて同時に主記憶に入れることができる．カード読取り機は平均1,000枚／分で入力し，各印刷機は平均1,000行／分で出力する．

ジョブの実行時間は，多重プログラミングを行なわずに単独で実行し，すべての入出力要求には時間がかからないとしたときの値である．実行時間は，ジョブ処理に必要なCPU時間を示している．

多重プログラミングを行なわないときの合計経過時間は次の値に等しい：

入力時間＋実行時間＋出力時間

たとえば，ジョブが単独に実行しているとすると合計の時間は次のように計算される：

入力時間＝100/1,000＝0.1分
実行時間＝3分
出力時間＝2,000/1,000＝2.0分
合計時間＝5.1分

上の計算は，多重プログラミングが行なわれておらず，入出力は同時実行

されていないと仮定していた（すなわち，入出力要求の処理中に CPU は停止している）：

1. スプーリングもバッファリングも行なわないとしたら，これらのジョブを実行するのに最もよい方法は何か．5つのジョブ全部の処理時間は幾らか．平均の返却時間（ジョブの最初のカードが読込まれているからそのジョブの最後の行が印刷されるまでの時間）は幾らか．
2. スプーリングは行なわないが，カード1枚分のバッファリングを行なうとして（プログラムでカードが要求される度に，次のカードの読込みを起動しておく），これらのジョブを実行するのに最適な方法は何か．合計の実行時間と平均返却時間を求めよ．
3. スプーリング(直接アクセス記憶装置を読み書きする時間を無視する）を行なうとして，これらのジョブを実行するのに最適な方法は何か．合計の実行時間と平均返却時間を求めよ．
4. 問3では，スプールが完全に終了するまではジョブの実行を開始することができなかった．日曜日の午前3時に1つのジョブが提出され，そのときには他のジョブが実行していなかったと仮定する．また，このジョブは5分間実行するジョブで 2,000 枚のカードを読込み，4,000 行印刷するものとする．
   - a　スプーリングやダブル・バッファリングを行なわないとして，このジョブが終了するまでにどれだけの時間がかかるか．
   - b　スプーリングは行なわないが，カード1枚分のバッファリングを行なうとしたら，このジョブが終了するまでにどれだけの時間がかかるか．
   - c　上述のスプーリング・システムを用いるとしたら，ジョブが終了するまでにどれだけの時間がかかるか．
   - d　スプーリング・システムを変更して，ジョブの入力が完了していなくても直接アクセス記憶装置からカードの読出しを開始でき，同様に，ジョブが終了する前に印刷出力を開始できるようにすると，このジョブが完了する時間はどれだけか．
5. 問4dのスプーリング・システムの組込み方法を簡単に述べよ（普通のスプーリング・システムの組込みとどのような点で異なるか）．
   - a　このシステムでは余分の同期が必要とされるか．
   - b　このスプーリングの技法を用いるとシステムの仕事量を逆に害するような場合が存在するか．

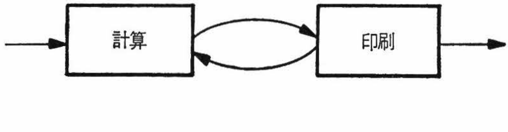
（ヒント：ジョブ・スケジューリングと予想しないジョブの到着について考察せよ.）

5.13 本章では論理レコードのブロッキングの原理とその利点を述べた．ブロッキングの主たる欠点は，大きな物理ブロックをバッファリングするために必要な主記憶容量である．大きな計算機で磁気テープを使用するときには，16Kバイトというブロック容量もめずらしいことではない．ダブル・バッファリング法を用いるときには，バッファとして32Kをとっておかねばならない．使用可能の記憶域が8K，磁気テープ1台，および印刷機1台を持つ IBM 360があったとして，バッファリング係数が100で120バイトの論理レコードをブロッキングした磁気テープ（12,000バイトの物理レコード）の内容を印刷することは可能か．可能とすればどのようにして行なうか．不可能ならば理由を示せ．

5.14 本章では，多重プログラミング・システムにおいて一度に複数の利用者のために，スプーリング・システムを用いて，仮想のカード読取り機と印刷機とを模擬することを論じた．基本的な処理は，ディスクやドラムを大容量のバッファ領域として使用，プログラムがこれらのバッファを読取り機や印刷機とみなすようにすることであった．全く同様の技法を用いると，同一のディスクを同時に使用して，あたかも複数台の別個のディスクが存在するかのごとく，仮想のディスク装置を模擬することが可能である．この方法は，別々のディスク装置を用いるようには速くできないかもしれないが，このような模擬によるオーバヘッドが是認されるような場合が存在するか．

5.15

1. 入力スプーリングのために磁気テープよりもディスクを使用するとどのような利益があるか．この利点は出力スプーリングにも当てはまるか．
2. 次図に示す構造のプログラムがあるとする：



プログラムではループを数千回繰返すものとする．プログラムは，多少

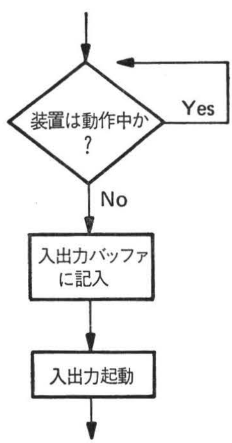
能率よくしようとして，印刷出力のためにダブル・バッファリング方式を用いている．1回に1行印刷される．

このようなプログラムが，スプーリングを行なうオペレーティング・システムの下で実行されると，オペレーティング・システムは入出力プログラミングを行なわなければならない．入出力に対して行なわねばならないことは何か．プログラムの実行速度にどのように影響するか．このようなプログラムが同時に複数個実行されるとしたら，システムの仕事量は増加するか．

3. たいていのミニコンピュータは単一利用者システムとして設計されている．したがって，各利用者は自分自身の入出力を行なうことが許されている（特権状態とか非特権状態というものはない)．各利用者は，自分で割込みを制御し，入出力などを処理する．

このようなシステムにカード読取り機，印刷機およびディスクが設置されている．このシステムを2人の利用者で多重プログラミングするように変更するとしたら，最も影響を受けない装置は何か．その理由も述べよ．

4. 問3で述べたシステムで，次のように入出力を起動するものとする：



第2の利用者が追加されると，入出力の流れ図をどのように変更しなければならないか．

（ヒント：3つの装置がすべて影響を受けるわけではない．）

5. このシステムでは，装置によってはオペレータの介入がなければ共用することが不可能である．どの装置か．その理由は何か．

5.16

1. スプーリングの利点を3つあげよ．それぞれを1文で説明せよ．
2. 通常，スプーリングは，カード読取り機，穿孔機，および印刷機だけに対して行なわれる．磁気テープに対するスプーリングの使用が望まれる理由を2つあげよ．どのような問題を解決することができるか．
3. 問2で述べた理由を考察して，磁気テープに対しては通常スプーリングを行なわない理由を述べよ．
4. スプーリングはすべての種類の入出力装置に適しているか．

5.17 入出力をスプーリングするのに2つの方法がある．1つは，印刷機や読取り機に対するチャネル・コマンド語を用いずに，直接ディスクのレコードにアクセスする GET/PUT ルーチを用意することである．他は，模擬を行なっている装置に対するチャネル・プログラムを受付けて，これを解釈する入出力起動ルーチンを用意することである．それぞれの方法の特徴について論じよ．

5.18 チャネルは，すでに論じたように，チャネル・コマンド語の中に実際の物理番地を必要とする．ハードウェアの変更なしに，入出力チャネルに対してページングを行なっている主記憶を用いると，入出力動作の処理中に関係のページが置換えの対象となる可能性が生じる．この問題の解決法として2つの可能性が存在する．

1. 仮想記憶の中のバッファを常駐化して，チャネル・コマンド語を実番地に変更する．すなわち，バッファとなっているページをページングのソフトウェアが置換えの対象としないようにする．
2. バッファをオペレーティング・システムの中のページングされない領域にとり，入出力が完了したとき，データを仮想記憶中のページに入れる．

問

a．方法1を組込むにはどのようにすればよいか．すなわち，どのデータベースやルーチンを変更せねばならないか．ページを常駐化することをどのようにして決定したらよいか．

b．方法1の場合で，バッファがページ境界にまたがっており，2つの

ページが実の記憶域で連続していないときには，どのようにして入出力が動作を行なうか．

c．方法2では，実の記憶域は，ページングを行なっている部分とオペレーティング・システムの領域とに分割される．利用者が入出力要求を出したときオペレーティング・システムにバッファ領域が残っていないとしたら，領域が解放されるのを待合せなければならないか，あるいは，オペレーティング・システムはページングを行なっている領域をバッファとして使用するように変更できるか．理由と，新たな問題点とを述べよ．

d．どちらの方式が優れているかを決定するには何を規準にして行なえばよいか．各方式の利点と欠点を1つずつあげよ．

e．チャネルが仮想番地のチャネル・コマンド語を処理できるとしたら，入出力処理は簡単となるか．理由と方法を述べよ．

5.19 読者は中規模な会社の給与計算システムを作成する仕事を受持っている．ほとんどのプログラミングがすみ，給与のマスタ・ファイルの物理的な書式を決定することだけが残っている．すでに決まっていることは，ファイルの論理的な構成は順編成とし，社会保障番号の順に並べ，1つのレコードは128バイトとすることである．ハードウェアは，IBM 3330型ディスク装置のついたIBM 370/165システムである．ディスク装置の特性は次に述べるとおりである．

各トラックに格納されるレコードはレコード間隙で分離されている．各シリンダにはトラックが19あり，同一のシリンダ内でアクセスするときには腕の移動が不要である．別のシリンダにアクセスするときには腕を移動させる必要がある．レコードの読出し時間は，腕の移動時間＋回転時間＋転送時間である．さらに，1つのトラックにおいて隣接のレコードを読出す時間はレコード間隙の長さ＋転送時間である．

1回転の時間＝16.7ミリ秒
平均の回転遅れ時間＝8.4ミリ秒
隣接トラックへの腕の移動時間＝10ミリ秒
腕の乱移動時間＝最大55ミリ秒，平均30ミリ秒
転送速度＝1.24マイクロ秒／バイト，
レコード間隙＝135バイト
トラック容量＝13,165バイト

1トラック当たりのレコード数＝13,165/(135＋d1)

ただし，d1はレコード長とする．

入出力起動命令が実行されてからディスク装置が接続されるまでの時間は無視してよい．

ソフトウェアは次の4つの基本機能を有しているとする：

1. 給与ファイルを更新して昨日の労働時間を示す．
2. 給与支払の小切手を印刷し，1週間に1回報告書を作成する．
3. 新しい従業員を追加する．
4. 退職者を取除く．

プログラムはすでにでき上っているので，動作時間がわかっている．印刷時間と読込み時間を無視する．更新プログラムは75マイクロ秒おきに入出力を要求し，ファイルの始端から終端への順で処理を行なう．支払小切手のプログラムは1ミリ秒おきに入出力を要求し，従業員の追加削除のプログラムは50マイクロ秒おきに入出力を要求する．

物理的なファイルの構成には3とおり考えられる．第1は純粋に順編成であり，シリンダ1，トラック1のレコード1から始まり，トラック2，トラック3，……トラック19と進み，次にシリンダ2，トラック1のレコード1と進むものである．第2の方法は，各レコードは情報の128バイトのほかに，次レコードの番地を納める6バイトを持つような連鎖状の構造をとる．レコードはディスクの領域の中にランダムに分布している．全体のレコードを読出すには，シリンダ1，トラック1のレコード1から開始して，このレコードの内容によって次のレコードを読出す．第3の方法は第2法を改良したものである．ファイルをシリンダ境界において分割し，1つのシリンダのレコードはすべて，主記憶中のシリンダ指標表の中の社会保障番号の範囲に含まれるようにするものである．しかし，シリンダの中ではレコードはランダムに並んでいる．この方式でレコードを求めるには，シリンダ指標表を調べてシリンダを求め，そのシリンダの第1レコードから必要とするレコードをたどって求める．

従業員が20,000人と仮定して，上述の3つの方法について次の問に答えよ．連鎖の方法を用いるときには，“あふれレコード”が各シリンダに存在すると仮定せよ．このレコードは連鎖して用いることが可能であり，従業員を追加するのに使用できる．

1. マスタ・ファイルのためにどれだけの領域が必要か．

2. 支払小切手をすべて印刷する時間は幾らか.
3. 1人の従業員を追加する時間は幾らか. 2人ならどうか.
4. 1人の従業員を削除する時間は幾らか. 2人ならどうか.
5. レコードをすべて更新する時間は幾らか.
6. シリンダにあふれレコードがなくなったら何が起こるか. そのための時間はどれだけかかるか.
7. 3つの方法の内どれがよいか. その理由は何か.

第6章

情報管理

6.1 はじめに

情報管理は，ちょうど図書館と同じように，システムに委託された情報の格納と検査を行なう．情報管理の基本機能は次に示すものである：

1. 各種の表を用いてシステム中のすべての情報の状態を把握する．主要な表はファイル•ディレクトリである．これはボリューム内容表（VTOC）と呼ばれることもある．これらの表は，システム中のすべての情報の名前，番地，およびアクセス権を納めている．
2. 情報をどこに，どのようにして格納するか，および誰に参照を認めるかを決定する方針を選択する．この方針を左右する要素は，必要とされる情報に対する2次記憶装置の有効利用，効率のよいアクセス，利用上の融通性，およびアクセス権と保護である．
3. 情報という資源の割付け．1つのプロセスに情報のアクセスを認めることに決定すると，割付けモジュールは所要の情報を求め，プロセスがアクセスできるようにし，適当なアクセス権を確立する．
4. 資源の解放．情報が不要になると，一時使用の表の欄やその他の資源を解放することができる．利用者が元の情報を更新したならば，他のプロセスでも使用できるように，情報の原本を更新する．

情報管理のモジュールは，総称して**ファイル・システム**と呼ばれることがある．情報管理は原理的には単純である．しかし，情報はオペレーティング・システムの最も重要な資源の1つであるが，恐らく現代のシステムでは最も粗末に扱われているものの1つでもある．

図1.9においては，情報管理モジュールをオペレーティング・システムの核

部分の外側の一層に入れていた．しかし，情報管理モジュールを完全に組込むことは全く複雑なことである．本章では，図 6.1 に示すように，情報管理モジュールをさらに幾つかの段階に分割して記述する．このような表現法は最初 Madnick (1969) が行なったものであるが，ファイル・システムの構成をよく表わしている．

ファイル・システムの各段階は，プログラム構造化の技法にしたがっている．各段階はそれ以下の段階にのみ依存し，下の段階のもののみを呼出す．ファイル・システムの設計の手法が階層的であるからといっても，オペレーティング・システムの他の部分が図 6.1 に示したように階層的に組込まれている必要は必ずしもない．

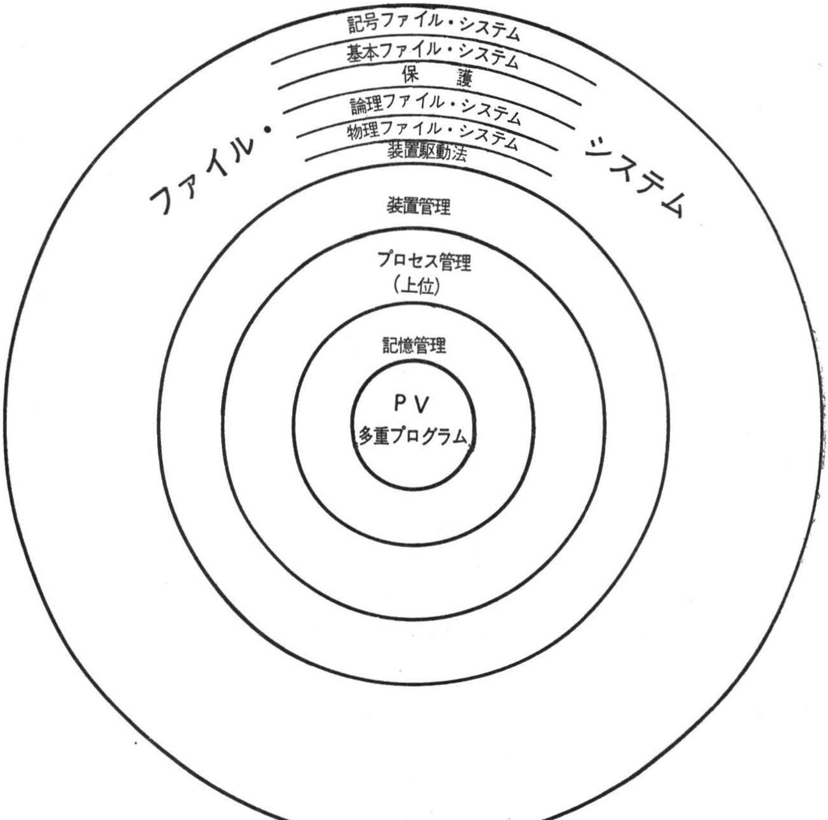


図 6.1 オペレーティング・システムの核

ファイル・システムは，恒久的（長期）な情報と一時的（短期）な情報の双方を都合よく管理しようとするものである．プログラマは，情報格納のための領域の割付けや物理的な格納のための書式，入出力のアクセスなどの問題から解放される．ファイル・システムの目標とすることの1つは，プログラマが情報の論理的な構造と処理の動作のみに注意を集中できるようにすることである．ファイル・システムは，利用者間での情報の共有と，許可されていないアクセスから情報を保護することをも可能とする．

本章では，ファイル・システムの基本原理だけを述べる．ファイル・システム（基本的な情報管理機構）とデータ管理システム，およびデータベース・システムの相違をはっきりさせておくことは有用である．

ファイル・システムは，単に情報の論理的な編成のみに関係する．ファイル・システムは，オペレーティング・システムの水準では構造化されず，意味づけのされていない情報の集合を取扱う．情報の個々の集合を**ファイル**とか**データ・セット**とかいう．

データ管理システムは，情報の構造化を行なうが意味の解釈は行なわない．たとえば，システムは各ファイルが一定の容量の項目に分割されていることを知っていて，これらの項目をプログラマが処理できるようにするが，情報の意味はわからない（IBM の IMS-IIはデータ管理システムである）．

データベース・システムは，データ項目の構造と解釈の両方に関係する．1つの例は，航空機の座席予約システムであり，利用者は“904 便に何人の乗客が乗るか”というようなことを問合せることができる．

本章は主としてファイル・システムについて論じる．データ管理およびデータベース・システムについては第6.7節で簡単に論じる．**ファイル**は，関連のある情報単位（**レコード**）の集合である．たとえば，在庫管理の応用では，伝票の1行は1つのデータ項目であり，1枚の伝票は1つのレコードであり，このようなレコードの集合がファイルである．

ファイル・システムは，システム・プログラムを管理する単純な機構から発展して，利用者とオペレーティング・システムに対して情報の格納，検索，および共用を行なわせるような一層複雑なものとなっている．このように，ファイル・システムは，利用者ジョブと同様に，オペレーティング・システムの他の成分から必要とされている．

本章では，まず，利用者が情報に対して出す簡単な要求を取上げて，この要求を満たすために必要となる手順を解析する．この簡単な例題からファイル・システムの一般的なモデルを構成し，これをもっと複雑で融通性に富むシステ

ムの設計に応用する.

6.2 簡単なファイル・システム

次の PL/I 流の文を処理するための手順を考えてみよう：

READ FILE(ETHEL) RECORD(4) INTO LOCATION (12000)

これは ETHEL という名前のファイルの第 4 レコードを主記憶の 12000 番地に読込む要求である. この要求はファイル・システムの要素を呼出す.

この例では, ファイル ETHEL は 図 6.2 に示したようにディスクに格納されているものとする（記憶装置の詳細については第 5 章を参照のこと）. ファイル ETHEL は斜線の部分に当り, 7 つの論理レコードから構成されている. 各論理レコードの長さは 500 バイトである. ディスクは, 1,000 バイトの物理レコード16個から構成されている. （記憶装置の物理的な単位を**ブロック**と呼び, プログラマが処理の対象とする**レコード**と呼ばれる論理的な単位と区別する. この関係は PL/I のデータ構造と同様である.）1 つの物理ブロックには, ファイル ETHEL の論理レコードを 2 つ格納することができる. 物理的な 2 成分の番地（トラック番号とレコード番号）を用いる代わりに, 図 6.2 に示すように, 各物理ブロックに一意的な番号を割当てることにする.

6.2.1 ファイル・ディレクトリ

各ファイルの状態に関する情報, たとえば, 名前や番地など, は表の中で管理する必要がある. 前述したように, この表は, 多くのシステムでは**ファイル・ディレクトリ**とか**ボリューム内容表**とか呼ばれている. 最も有名なのは IBM OS/360 のものである.

ボリューム内容表という名前は, 記憶装置の媒体を“情報の入れ物”, すなわち, **ボリューム**とみなすことに由来する. 記憶装置とボリュームの区別は, 取外し可能なボリューム（たとえば, ディスク・パックや磁気テープのリール）が使用されるときに重要となる. これらのボリュームは, 物理的に元の記憶装置から同種の別の記憶装置に移動させることができる. 時として, 別の計算機システムのことすらある.

図 6.3 はファイル・ディレクトリの例を示している. 各欄には, ファイルを探してアクセスするために必要な情報を収容している. プログラマからの観点

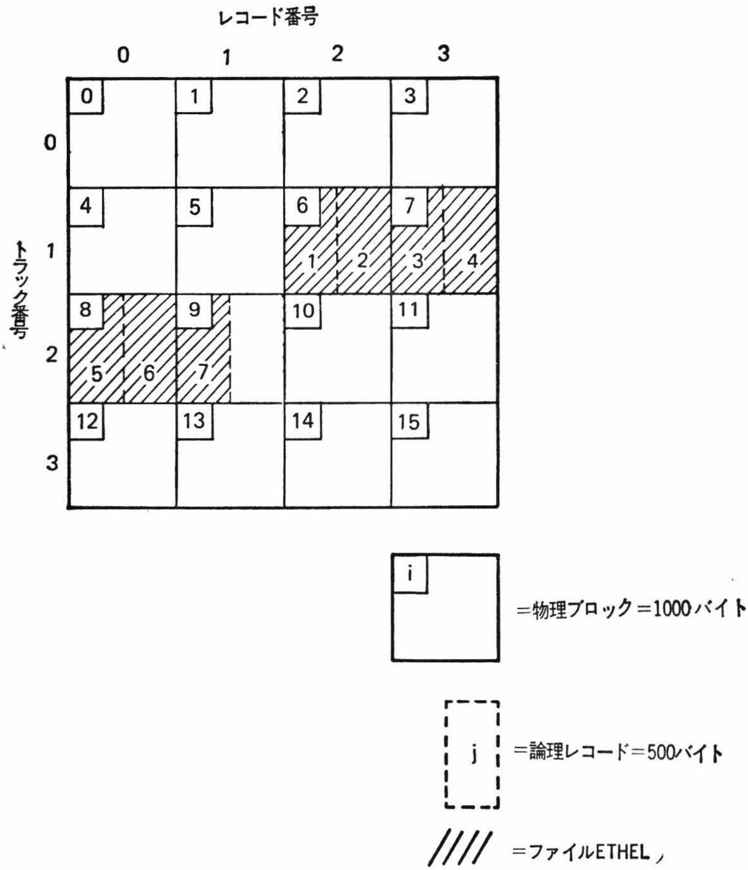


図 6.2 ファイルの物理的な格納

であるファイルの論理的な編成は，論理レコード長およびレコード数によって示される．（本節では，ファイルの中のレコードはすべて同一の長さであると仮定する．別のファイル編成は，後で第6.7節で述べる．）

記憶装置の中のファイルの物理番地は，情報の先頭ブロック番号，論理レコ

欄番号	名前	論理レコード長	論理レコード数	第1物理ブロックの番地	保護とアクセス制御
1	MARILYN	80	10	2	全員
2	空き	1000	3	3	(空き)
3	ETHEL	500	7	6	読出しのみ
4	JOHN	100	30	12	MADNICKには読出し DONOVANには読出し/書込み
5	空き	1000	2	10	(空き)
6	空き	1000	1	15	(空き)

図 6.3 ファイル・ディレクトリの例

ード長，および番号から決定することができる．たとえば，ファイルETHELは500バイトのレコード7つから構成され，3,500バイトの記憶域が必要である．物理ブロックの長さは1,000バイトであるから，ファイル ETHEL を収容するのに4つのブロックが必要である．これらのブロックの最初は，ディレクトリに示されているように，6である．したがって，図6.2に示したように，ETHEL は物理ブロック6，7，8および9を使用している．

重要なことは，論理的な構造と物理的な構造を区別することである．利用者から見たとき，ファイル ETHEL は論理的に順編成ファイルの構造をとっている．ファイルは，1番から7番までの7つのレコードから構成されている．また，このファイルは物理的にも順ファイルとして格納されている．すなわち，レコードは，物理的に連続したブロックの中に，隣接して格納されている．しかし，ETHEL の物理的な構造は必ずしも順編成である必要はない．ETHEL のブロックは2次記憶に散らばって入ることも可能である．このときには，論理レコードと物理レコードとの対応関係は複雑となる．別の構造の例については後で第6.8節で述べる．

図 6.3 のファイル・ディレクトリには6つの欄があげられている．3つ(MARILYN, ETHEL, および JOHN)は実在の利用者ファイルに対応する．他の3つの欄は，現在未使用となっている記憶域を管理する目的でのみ必要と

される．このようにして，ファイル・ディレクトリは，すべての物理ブロックの使用状況を示している．（註：ブロック0および1は，後述する特別の目的のためにとってある．）

6.2.2 実行すべき手順

図6.4は次のような要求を満たすために実行しなければならない手順を示している：

READ FILE(ETHEL) RECORD(4) INTO LOCATION(12000)

これらの手順は図6.3で示したファイル・ディレクトリの例を利用している．

段階番号	動 作
段階1	ファイル・ディレクトリを探してETHELの欄を求める(第3欄)
段階2	ETHELに関する情報をファイル・ディレクトリの欄から求める： 論理レコード長，第1物理ブロック番地，保護とアクセス制御の情報．
段階3	保護の情報に基づいてETHELをアクセスしようとする処理要求を許すかどうかを決定する．
段階4	指定された論理レコード番号(ETHELのレコード4)を論理バイト番地に変換する． 論理バイト番地＝(レコード番号－1)×(レコード長) ＝(4－1)×(500)＝1500
段階5	次に論理バイト番地を物理バイト番地に変換する．論理バイト番地1500はETHELの第2物理ブロック(1000バイト)の中のバイト500に対応し，この物理ブロックは第7ブロックに当る．
段階6	物理ブロック7を主記憶の1000バイトのバッファに読込む． これは第2および第5章で述べたような入出力プログラムによって実行される．
段階7	レコード4がバッファ領域から取出され，利用者の領域の12000番地に移される． すなわち，バッファの番地500から999までが主記憶の番地12000から12499に移される．

図 6.4 READ 要求の処理の段階

読者は手順を1つずつ追って，アルゴリズムが正しいことを確かめてほしい．

6.3　ファイル・システムの一般的なモデル

本節では，ファイル・システムの一般的なモデルを紹介する（Madnick, 1969）．図 6.5 にはファイル・システムの主要な要素を示している．これらの要素の大半は，図 6.4 の手順を一般化したものに対応している．

第1章で紹介した考え方に従って，このファイル・システムの要素を階層構造に編成する．階層の各段階は，それより下の段階にのみ依存する．この考え方を用いると，組織的なデバッグ手順をとることがより容易になる．たとえば，最も下の段階のデバックがすむと，これより上の段階での変更には影響されない．デバッグは最も下から最も上のレベルへの順で行なうことができる．

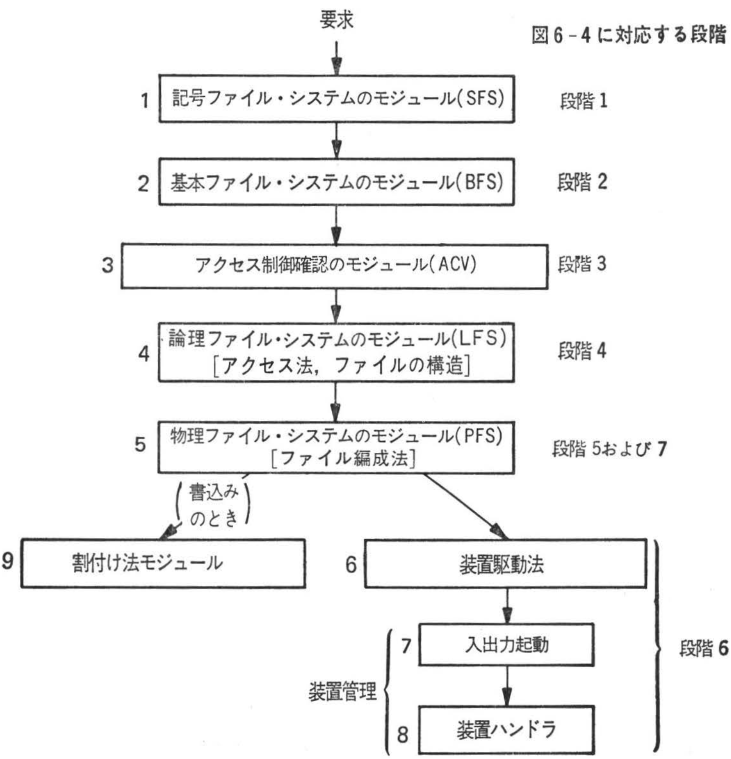


図 6.5　ファイル・システムの階層的モデル

本章で述べるような詳細（たとえば，ファイル・ディレクトリの書式やファイル・マップ）に関しては，システムによって非常に異なっているが，基本的な構成はたいていのファイル・システムについて共通である．たとえば，ファイル・システムによっては，図6.5のモジュールの中で1つに併合されるものもあるし，2つ以上に分割されることもある．また，場合によっては組込まれないこともある．それでも，基本的な構造は同じである．

6.3.1 ファイル・ディレクトリの管理

ファイル・システムの要素について論じる前に，ファイル・ディレクトリに関して，

> “ファイル・ディレクトリの項目はどのようにして作られ，記入されるか．”
> “ファイル・ディレクトリはどこに格納されるか．”
> “要求が出される都度ディレクトリ全体を探索しなければならないか．”

などの問に答えておくことが重要である．

IBM 基本オペレーティング・システム BOS/360 などの基本的なシステムでは，プログラマは空き領域の管理とファイル・ディレクトリの管理を次のような制御カードによって行なわなければならない：

```
CREATE   ETHEL, RECSIZE=500, NUMRECS=7,
                LOCATION=6
DELETE   JOHN
```

CREATE 命令はファイル・ディレクトリに新しい項目を追加し，DELETE 命令は古い項目を削除する．

ファイル・ディレクトリ全体をつねに主記憶に納めておくとしたら，このために大容量の記憶域が必要である．より一般的な方法はファイル・ディレクトリを1つのファイルとして扱い，記憶ボリュームに収容することである．さらに，ファイル・ディレクトリがそのボリュームに格納されていると，ファイルとそれに対応するファイル・ディレクトリを収容しているボリューム（磁気テープのリールやディスク・パックなど）を物理的に移動させることにより，ファイルを容易に他のシステムに移すことができる．

ファイル・ディレクトリがボリュームに格納されているとき，ファイルをアクセスするたびにディレクトリを探索すると非常に時間がかかる．ディレクト

リには数千もの項目が納められていることもあり，非常に大きなものであるが，通常一時には少数のファイルだけが使用される．したがって，使用中のファイルに関する項目だけを主記憶に複写しておくと，それ以後の探索時間を大幅に減少させることができる．多くのファイル・システムでは次の2つの特別なコマンドを備えている：

OPEN　主記憶にファイル・ディレクトリの項目を複写する．
CLOSE　この項目は主記憶の中に置く必要がなくなったことを示す．

同様にして，ファイル・ディレクトリの項目がどこに置かれているかによってファイルが**開かれている**（オープン）とか**閉ざされている**（クローズ）とかいう．

6.3.2 記号ファイル・システム

第1の処理手順は**記号ファイル・システム**（SFS）と呼ばれる．このシステムの呼出しは次のようにして行なわれる：

CALL　SFS(動作，ファイル名，レコード番号，主記憶番地)

その例は次のようになる：

CALL　SFS(READ，"ETHEL"，4，12000)

記号ファイル・システムは，ファイル名を用いてファイル・ディレクトリの項目の格納場所を見つける．この機能は図6.4の手順1に対応する．

本節の例では，ファイル名とファイルには1対1の対応があると仮定しておく．もっと進歩したファイル・システムでは，同名異物のファイルや，同一ファイルを別名で参照する機能が必要とされる．この機能を組込むには，図6.3のファイル・ディレクトリを，図6.6に示したように，記号ファイル・ディレクトリ（SFD）と基本ファイル・ディレクトリ（BFD）の2つに分割する．記号ファイル・システムでは，要求されたファイルをSFDで探して識別番号（ID）を決定する（図6.6参照）．BFDでは，ファイル名ではなく，このIDを使用して項目が探される．基本ファイル・システム（BFS）は，識別番号を使用して次のようにファイルの処理を行なう：

CALL　BFS(READ，5，4，12000)
ただし，5はETHELのIDである．

通常，SFDは2次記憶装置に格納されているので（図6.6bのID 2を参

	名前	識別符号(ID)
1	MARILYN	3
2	ETHEL	5
3	JOHN	6

←16バイト→ ←4バイト→
← 各欄20バイト →

(a) 記号ファイル・ディレクトリの例

ID	論理レコード長	論理レコード数	第1物理ブロックの番地	保護とアクセス制御
1	—	—	0	(基本ファイル・ディレクトリ)
2	20	3	1	(記号ファイル・ディレクトリ)
3	80	10	2	全員
4	1000	3	3	(空き)
5	500	7	6	読出しのみ
6	100	30	12	MADNICKには読出し DONOVANには読出し/書込み
7	1000	2	10	(空き)
8	1000	1	15	(空き)

(b) 基本ファイル・ディレクトリの例

図 6.6 記号ファイル・ディレクトリと基本ファイル・ディレクトリの例

照)，使用中（**アクティヴ**とか**開かれたファイル**と呼ばれる）のファイルのための項目は主記憶に複写される．このように複写すると，2次記憶装置の中の同じ項目に対して入出力を何回も行なわなくてすむ．このようなアクティヴ・ファイルの項目を並べた表をアクティヴ名前表（ANT）という．

要約すると，記号ファイル・システムは次のものの管理および動作を行なう：

1. 記号ファイル・ディレクトリ（SFD）
2. アクティヴ名前表（ANT）
3. 基本ファイル・システム（BFS）との通信

6.3.3 基本ファイル・システム

第2の処理手順は**基本ファイル・システム**（BFS）と呼ばれる．呼出しの形式は次のとおりである：

CALL BFS(動作，ファイル識別番号，レコード番号，主記憶番地)

その例は，第6.3.2節で示したように，次のようになる：

CALL BFS(READ，5，4，12000)

基本ファイル・システムでは，識別番号（ID）を使用して，BFD中のファイル項目を探し（図**6.6b**参照），この項目を主記憶に複写する．この機能は，図6.4の手順に対応する．

SFDに対して特定のIDを恒久的に割当てておくと（たとえば，図**6.6b**のID 2)，SFSは，所要の名前をディレクトリ内で探してIDを決定するために，BFDの機能を利用することができる．たとえば，次のPL/Iセグメントを用いるとSFDを探索することができる：

```
          DECLARE 1   SFD_ENTRY,
                      2 NAME  CHARACTER (16),
                      2 ID    FIXED BINARY (31);
          DO     I = 1 TO 3;
                 CALL  BFS (READ, 2, I, SFD_ENTRY);
                 IF  SFD_ENTRY.NAME = DESIRED_NAME
                 THEN GO TO FOUND;
                 END;
FOUND:           DESIRED_ID = SFD_ENTRY.ID;
```

アクティヴ・ファイルのBFD項目は，アクティヴ・ファイル表(AFT)に格納される．このようにすると，同一の項目に対してBFDを2回以上アクセスしなくてもよい．

次の手順であるアクセス制御確認（ACV）では，ファイルのIDの代わりにAFTの項目を用いて，次のような呼出しに対する処理を行なう．例では，AFTの項目2はID 5（ファイル ETHEL）のファイルの情報を収容しているものとする．

CALL　ACV(READ, 2, 4, 12000)

要約すると，基本ファイル・システム（BFS）は次のものの管理および処理を行なう：

1. 基本ファイル・ディレクトリ（BFD）
2. アクティヴ・ファイル表（AFT）
3. アクセス制御確認モジュール（ACV）との通信

第6.2節のようなファイル・システムでは，SFSとBFSとを組合せることはきわめて容易である．後節では，これら2つの論理的な分割がはるかに重要となるような，機能の付加を行なう．

6.3.4 アクセス制御確認

第3の処理手順はアクセス制御確認（ACV）と呼ばれる．呼出しの形式は次に示したものである：

CALL　ACV(動作，AFT 項目，レコード番号，主記憶番地)

呼出しの例は，第6.3.3節で示したように，次の形式で行なわれる：

CALL　ACV(READ, 2, 4, 12000)

ACVは，基本ファイル・システムと論理ファイル・システムとの間での検問を行なう．すなわち，要求された動作（READ，WRITEなど）が，AFTに示されたアクセス許可に含まれるかどうかを比較する．そのアクセスが許可されない場合には，誤りの状態であるので，ファイル・システムに対する要求の処理を行なわない．そのアクセスが許可される場合には，制御は直ちに論理ファイル・システムに渡される．この機能は図6.4の手順3に対応する．

6.3.5 論理ファイル・システム

第4の処理手順は論理ファイル・システム（LFS）と呼ばれる．呼出しは次の形式で行なわれ，第6.3.3節および第6.3.4節で示したACVの呼出しと同一の形式である：

CALL　LFS(動作，AFT 項目，レコード番号，主記憶番地)

呼出しの例は次のようになる：

CALL LFS(READ, 2, 4, 12000)

論理ファイル・システムは，論理レコードに対する要求を論理バイト列に対する要求に変換する．すなわち，ファイルは物理ファイル・システムにおいては，記憶の書式に関係なく，一連のバイト列として取扱われる．本章の例のように固定長レコードの場合には，AFT 項目の中のレコード長を用いて必要とされる変換を行なうことができる．すなわち，次の関係が成立する：

論理バイト番地＝（レコード番号－1）×レコード長
論理バイト長＝レコード長

この機能は図 6.4 の手順 4 に対応する．

論理バイト列番地と長さは，次の物理ファイル・システムの段階でレコード番号の代わりに使用される．物理ファイル・システム（PFS）は次のような呼出しの処理を行なう．

CALL PFS(READ, 2, 1500, 500, 12000)

BFD のファイルに対して特定の ID（図 6.6 b の ID 1）と AFT の項目番号（AFT の項目 1）とを恒久的に割付けておくことにより，基本ファイル・システムは，BFD の項目を読み書きするために，論理ファイル・システムを呼出すことができる．（システムが最初起動されるときや再起動されるとき，BFD の項目 1 を AFT の項目 1 に記入するには，特別の手続きを使用しなければならない．）

要約すると，論理ファイル・システム（LFS）は次の処理を行なう：

1. レコードの要求をバイト列の要求に変換すること
2. 物理ファイル・システム（PFS）との通信

可変長レコードなどのより複雑な論理レコードの編成の例は後節に示される．

6.3.6 物理ファイル・システム

第 5 の処理手順は物理ファイル・システム（PFS）と呼ばれる．このシステムの呼出しは，第 6.3.5 節に示したように，次の形式で行なわれる：

CALL　PFS(動作，AFT項目，バイト番地，バイト長，主記憶番地)

呼出しの例は次のようになる：

CALL　PFS (READ，2，1500，500，12000)

物理ファイル・システムでは，このバイト番地とAFT項目とを使用して，所要のバイト列を格納している物理ブロックを決定する．このブロックは装置駆動モジュールの機能を用いて主記憶中のバッファに読込まれ，バイト列が抽出されて利用者のバッファ領域に移される．この機能は図6.4の手順5および7に対応する．

図6.2のように単純に連続した編成の場合には，論理バイト番地から物理ブロック番号への写像は次のようにして行なうことができる：

$$物理ブロック番号=\frac{論理バイト番地}{物理ブッロク容量}+先頭ブロックの番地$$

論理バイト番地1500のバイト列に対する要求に対しては，物理ブロック番号は次のようになり，ファイル ETHEL のレコード4を納めているブロックは7となる（図6.2参照）：

$$物理ブロック番号=\frac{1500}{1000}+6=1+6=7$$

物理ブロック内でのバイト列の番地は次式によって決定される：

$$物理ブロック・オフセット=剰余\left[\frac{論理バイト番地}{物理ブロック容量}\right]$$

例の場合は次のようになる：

$$物理ブロック・オフセット=剰余\left[\frac{1500}{1000}\right]=500$$

したがって，バイト列は，図6.2からわかるように，物理ブロックの内のオフセットが500の所から始まっている．

このような計算を行なうには，物理ファイル・システムに対して写像の関数および各記憶装置で使用されている物理ブロック容量が示されなければならない．これらの値がすべての装置について同一であれば，その値を物理ファイル・システムのルーチンに組込んでしまうこともできる．しかし変換は，これらの値は装置の種類によって変化する（大容量ファイルと小容量のディスクの取

扱いは異なっている). したがって, この情報は, 通常, 記憶ボリューム自体に格納されていて, システムが最初起動するとき, 物理構成表 (POT) に読込まれる.

WRITE 要求の処理を行なおうとしたとき, これに対応する物理ブロックがまだ割付けられていないならば, 物理ファイル・システムは割付け法モジュール (ASM) を呼出して, 2次記憶装置の空きブロックの番地を求める. 第6.2節で示した単純なファイル・システムの場合には, すべてのブロックは, あらかじめ特別の CREATE 機能を用いて割付けられている.

要約すると, 物理ファイル・システム (PFS) は次のものの管理および処理を行なう:

1. 論理バイト列の要求を物理ブロック番号とオフセットに変換すること
2. 物理構成表 (POT)
3. 割付け法モジュールおよび装置駆動モジュールとの通信

後節では, 物理ファイル・システムに関係のあるもっと多くの物理的なシステム構成と効率の考察を行なう.

6.3.7 割付け法モジュール

割付け法モジュール (ASM) は, 各記憶装置の空きブロックを管理する. 呼出しは次の形式で行なわれる:

CALL ASM(POT 項目, ブロック数, 先頭ブロック)

呼出しの例は次のようになる:

CALL ASM(6, 4, FIRSTBLOCK)

ただし, POT 項目6は, ファイル ETHEL を格納しようとする記憶装置に対応し, FIRSTBLOCK は, 4ブロックの要求に対して先頭ブロックの番地が返却される変数である.

割付け法モジュールは, 第 6.3.1 節の CREATE 命令などの特別の準備手続きによって呼出されたり, 図6.5に示した物理ファイル・システムによって自動的に呼出されたりする.

図6.6b は, 空きブロックの番地が, これらを特別のファイルとして取扱う

ことによって，BFD の中でいかに記録されるかを示している．この情報を管理する別の技法については後節に示される．

6.3.8 装置駆動モジュール

装置駆動モジュール (DSM) は，物理ブロック番号を装置に必要とされる書式の番地（たとえば，図6.2に示したように，物理ブロック7＝トラック1，レコード3）に変換する．このモジュールは，装置の種類によって適当な入出力コマンドを作る．この機能は図6.4の手順6に対応する．これまでのモジュールはすべて装置の種類に独立であったが，このモジュールは装置に関係する．次に制御は入出力スケジューラに渡される．

6.3.9 入出力スケジューラと装置ハンドラ

これらのモジュールは，第5章で論じた装置管理のルーチンに対応する．これらは，所要の情報を格納している物理ブロックの読み書きをスケジュールして実行する．

6.3.10 ファイル・システムのモジュール間の呼出しと復帰

物理的な入出力が完了すると，制御は装置駆動モジュールに復帰する．装置駆動モジュールは，正常完了であることを確かめ制御を物理ファイル・システムに返却する．物理ファイル・システムは，バッファから所要の情報を取出して要求の番地に格納する．次に“成功”の符号がファイル・システムのその他のモジュールをとおして返却される．

最上位の段階に直接復帰しない理由は何であろうか．その理由は2つある：

1. 各段階で誤りが検出されることがあり，適当な誤り符号がその段階から返却される必要がある（たとえば，論理ファイル・システムでは，ファイル領域を越えてアクセスしようとすることが検出できる）．
2. 各段階はそれ以下の段階を複数回呼出すことがある（たとえば，記号ファイル・システムでは，SFD の中の項目を読出すために，基本ファイル・システムを何回か呼出す）．

図6.5の各段階の機能は確認である．これは，各要求の正当性を確認することである．記号ファイル・システムでは，ファイルが実在するかどうかを検査する．アクセス制御確認では，そのアクセスが許されているかどうかを検査す

る．論理ファイル・システムでは，要求された論理番地がファイルの中にあるかどうかを検査する．装置駆動モジュールでは，その装置が実在するかどうかを検査する．

6.3.11 簡単なファイル・システムの設計で不備な点

本節で述べた簡単なファイル・システムで行なった仮定，効率の悪い点，および不適当な点をここで要約しておこう：

1. **記号ファイル・システム**

   各ファイルは単一の，しかも一意的なファイル名を持っていると仮定した．

2. **アクセス制御確認**

   ファイルはどのようにして共用されるか．異なる目的の利用に対して，同一のファイルを異なるアクセス権で保護するにはどのようにすればよいか．

3. **論理ファイル・システム**

   各ファイルは固定長のレコードから構成され，順編成であると仮定した．可変長のレコードはどのように取扱われるか．別のファイル構成法は存在するか．利用者にとってもっと融通のきく方法があるか．

4. **物理ファイル・システム**

   順編成以外の物理的な構成法が存在するか．それらは領域の使用率の点でもっと能率がよいか．アクセスの仕方についてはどうか．本節で1つの要求が出される度にディスクに対するアクセスが行なわれるものと仮定した．要求

   READ ETHEL RECORD(4)

   の次に要求

   READ ETHEL RECORD(3)

   が出されたときどうなるか．レコード3はすでにバッファの中に入っているのではないか．

5. **割付け法**

   ファイルは2次記憶装置にどのように格納されるか．ファイルが削除されたとき断片化の問題が存在するか．いい換えると，2次記憶は効率よく使用されているか．

6. **装置駆動モジュール**

このモジュールは，情報が格納されている装置の特性により効率よく適合させて，要求の組換えを行なうことができるか．

以下の節では，これらの段階に対して，より一般的で，利用者に一層の融通性を与え，システムの性能を向上させる技法を探究しよう．

6.4 記号ファイル・システム

記号ファイル・システム（SFS）の機能は，利用者が行なう記号による参照を識別番号（ID）に写像することである．本節では，第6.3節で述べたよりも高度の次のような機能を実現する方式を考える：

1. **同一のファイルに複数の名前をつけること**
   この機能はしばしば**別名**と呼ばれ，別々のプログラムで同一のファイルに別の名前を使用しているときに有用である．利用者は，異なった状況で1つのファイルに一層意味のはっきりした名前をつけることもできる．
2. **ファイルの共用**
   この機能は**リンクされた**ファイルと呼ばれることもある．この機能によって，物理的に同一のファイルに複数の利用者がアクセス可能となり，一層融通性を与え，しかも，しばしばシステムの性能を大きく向上させる．その理由は，主記憶中のバッファも2次記憶の領域も共用可能となるからである．たとえば，数人の利用者が同一のファイルをアクセスしていても，そのファイルの写しは1つしか必要でない．たとえば，M.I.T. の CTSS タイムシェアリング・システムの初期のシステム（第9章参照）は，ファイルの共用が可能でなかったので能率が悪かった．このシステムには DOCTOR という名前の精神病医を模擬する非常によく使用されるプログラムがあった．システムの利用者の半数以上がこのプログラムの写しを別個に持っていた．このプログラムは大容量であったので，2次記憶装置の半分以上がDOCTORの写しで占められることもあった．
3. **別のファイルに同一の名前をつけること**
   多数の利用者がファイルを作成しそれを使用するので，2人の別のプログラマが1つのファイルにたまたま同じ名前（たとえば，TEST）をつけることもある．したがって，1人の利用者が次のような制御カード

を使用したとすると，他のプログラマのファイル TEST を削除してしまうかもしれない：

DELETE TEST/* 旧ファイルを削除 */
CREATE TEST/* 新ファイルを作成 */

同様に，READ および WRITE 要求も２人のプログラマに対して同一のファイルが使用されるであろう．名前の重複を避けるために，人手でファイル名を登録させて管理することは，時間がかかるうえに上手な解決法とはいえない．

4. **ファイルをグループに再分割すること**

１人の利用者（あるいは利用者の集団）は多数のファイルを使用するかもしれない．これらのファイルは，論理的には別個の計画や仕事に所属していることがある．これらのファイルを組織的に管理するには，互に関連するファイルをグループにまとめると都合がよい．このようなグループのことを，しばしば，**ライブラリ**と呼ぶ（IBM の呼び方では**分割編成データ・セット**）．

記号ファイル名は，利用者にとってファイルのことを記憶したり参照したりするのに都合がよいが，前に示した単純なファイル・ディレクトリ方式には幾つか不備な点がある．したがって，上述したような機能を追加するための，一層融通性のあるファイルの名前づけ方式を示そう．

6.4.1 ディレクトリ・ファイル

図 6.6 ではファイル・ディレクトリを，記号ファイル・ディレクトリ（SFD）と基本ファイル・ディレクトリ（BFD）の２つに分割していた．SFD はファイルの名前づけに関係し，BFD は物理ファイルに関係する．この機構を２つの方法で一般化することができる：

1. 複数の記号ディレクトリを用いる．
2. 記号ディレクトリをファイルとして取扱う．

名前づけによる混乱を解消したり少なくしたりするために，各プログラマあるいはプログラマの集団に別個の SFD を割当てることができる．このようにすると，各プログラマ用の SFD のための BFD 項目を規定する主ディレクトリが必要となる．

主ディレクトリは一意的な識別番号（ID）を持っている．すなわち，図 6.6

bでは BFD の項目2に対するファイルがつねに主ディレクトリのために使用される．このようにしておくと，記号ファイル・システムは，基本ファイル・システムを介して主ディレクトリにアクセスすることができる．

6.4.2 階層的ファイル構造の例

図 6.7 には，7つのデータ・ファイルと4つの SFD ファイルおよび BFD から成る単純な階層的ファイル構造の詳細を示している． BFD の第1項目は BFD 自体を示し，第2項目はつねに主ディレクトリに割当てられる．
この項目は，すべてのファイルと同様に，レコード長などを納めている．

主ディレクトリのファイルは，各プログラマの記号ファイル・ディレクトリのための項目を納めている．例では，DONOVAN と MADNICK の2人のプログラマが入っている．各プログラマのディレクトリには，そのプログラマが

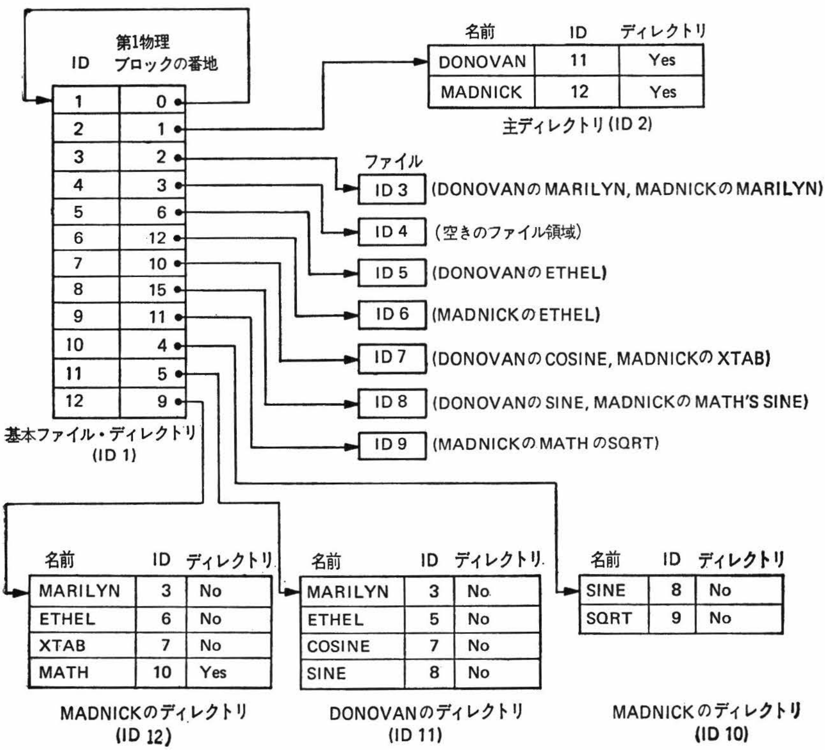


図 6.7 階層的なファイル構造の詳細

参照できるファイルすべてに対して，それを示す項目が入っている．DONOVAN は，MARILYN, ETHEL, COSINE, および SINE の 4 つのファイルを持ち，MADNICK は，MARILYN, ETHEL, XTAB, および MATH の 4 つのファイルを持っている．さらに MATH 自体は記号ファイル・ディレクトリであり，SINE および SQRT の 2 つのファイルに対する SFD 項目を納めている．

ファイル MARILYN は，2 人のプログラマにとって同一のファイル（すなわち，同一の ID）であることに注意してほしい．これは，しばしば，**リンクされたファイル**と呼ばれる．一方，DONOVAN のファイル ETHEL は，MADNICK のファイル ETHEL と同一ではない（すなわち，ID が異なる）．この例では，このほか幾つかの融通性のある機能が示されている．(DONOVAN のファイル COSINE と MADNICK のファイル XTAB は，実は，同一のファイルであることに注意してほしい．）読者は図 6.7 をよく調べてこれらのことを確認してほしい．

上述したような**階層的なファイル構造**は，しばしば，図 6.8 に示したように

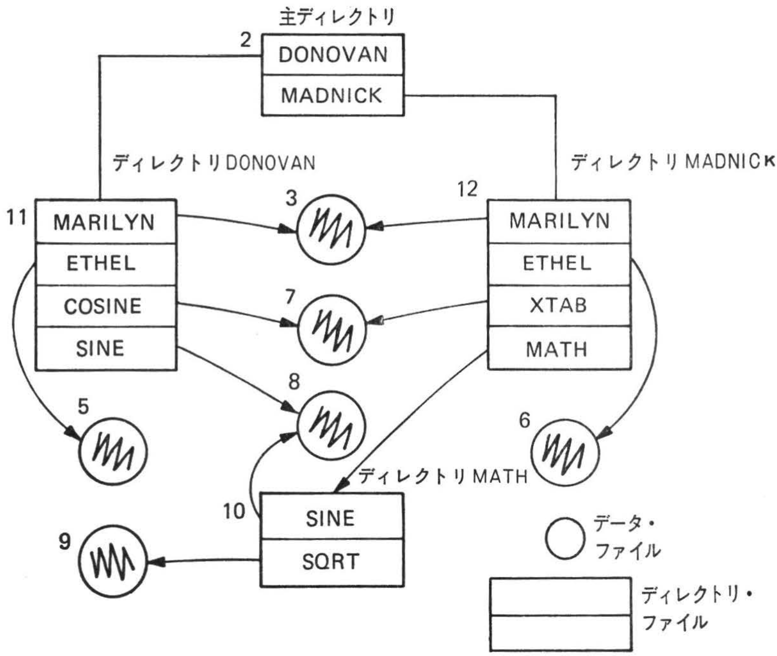


図 6.8 階層的なファイル構造の要約

樹構造で表現される．幾つかのファイル•システムでは(たとえば，MULTICS)，プログラマは多段階の副ディレクトリ——しばしば個人ライブラリとも呼ばれる——を作ることができ，その結果その構造は，図6.7および図6.8に示したMATH ディレクトリのように，幾つかの層から構成されることとなる．

6.5 基本ファイル・システム

基本ファイル・システムが呼出されたとき，要求のファイルの一意的な ID が与えられる．基本ファイル・システムは，まず，アクティヴ・ファイル表(AFT)を探索して，ファイルがすでに“開かれている”かどうかを調べる．ファイルが開かれていない場合には，基本ファイル•システムは BFD を探して，要求されたファイルの項目を求める．次に，この情報を AFT に格納する．

ファイルに割当てられている ID が BFD の項目番号と一致している場合には，BFD の項目を求めるのに“探索”が不要である．

利用者がファイルを抹消しようとするときなど（たとえば，DELETE DONOVAN'S ETHEL）の場合には，このファイルが2つ以上の SFD 項目として出現しているかどうかを知ることが必要である．これを行なうには，その BFD 項目を指している SFD 項目の個数を数えて，その値を BFD 項目に格納しておけばよい．リンクが行なわれたり別名が作られたりしたときには，この値が増加される．リンクや別名が削除されるときには，この値が減ぜられる．（ファイル自体が抹消されるときには，SFS は，対応する SFD 項目をすべて探して削除しなければならない．）

6.6 アクセス制御の確認

アクセスの制御と情報の共有とは相互に関連づけて考慮しなければならない．たとえば利用者は，自分だけしか入れない施錠した金庫室に全計算機システムを置いて情報を保護することも可能である．政府の計算システムの中には，3台の計算機システムを用いて，このような方法をとっているものがある．1つは最高機密の情報を扱い，第2は秘密情報を扱い，第3は重要事項を扱う．しかし，通常はこのような極端な形式の保護は好ましくない．

制御された共有機能があると，利用者に便利であるうえに，次のような幾つかの利点が存在する：

1. ファイルの複写を必要としない．たとえば，1つのシステムで 128K のプログラムを共有していてこれがディスクに入っていたとする．共有が行なわれなくなり，たとえば，50の個人ファイルが作られるとするとディスク領域を 6.2M バイト使用する．
2. データ処理の同期を行なうことができる．2つのプログラムが銀行の口座勘定を更新しているとすると，常時，同じファイルを参照することが望まれる．（更新処理を整合させるために，第4章で論じたような適当なインターロックが必要となることに注意してほしい.）
3. システムの性能を改善する．たとえば，主記憶中で情報が共用されると，入出力動作が減少する．

本節では，アクセス制御を管理して実施する3つの技法——アクセス制御行列，合言葉，および暗号法——について述べる．

6.6.1 アクセス制御行列およびアクセス制御表

たいていのシステムでは，ジョブを実行している利用者を識別する機構が備わっている．この機構は，計算機利用料金を正しく課すために必要である．この識別は，計算機オペレータの目で行なわれることもあるし，ジョブの一部として提出される特別の識別符号（電話で利用するときのクレジット・カード番号と同様）のこともある．本書ではこの機構についてこれ以上詳しくは述べないが，とにかく，利用者は識別可能であると仮定しておく．

アクセス制御行列は2次元行列である．図 6.9 に示すように，1つの軸にはすべての計算機利用者を示し，もう1つの軸にはシステム中のすべてのファイルを示している．行列の各欄は，対応する利用者の対応するファイルに対するアクセス権を示している（たとえば，DONOVAN はファイル PAYROLL を読出すことができ，ファイル STOCKPRICE を読み書きすることができる．しかし，MADNICK はファイル STOCKPRICE の読出ししかできない)．アクセス制御確認モジュールは，利用者のアクセス要求とその利用者に対して許されたファイルへのアクセス権を比較する．もし適合しなければ，そのアクセスは認められない．

図 6.9 のアクセス制御行列は，原理的には単純であるが，組込みに際しては幾つかの問題がある．たとえば，M.I.T. では約 5,000 人の利用者が登録されており，約30,000のオンライン・ファイルが存在する．このための2次元行列は，5,000×30,000＝150,000,000欄が必要である．図 6.9 とは論理的に等価で

セグメント＼利用者	Donovan	Madnick	Ziering	悪者	・・・
SQRT	E	RWE	E	E	
PL/I	RWE	E	E	なし	
株価	RW	R	R	R	
給与	R	RW	R	なし	
・					
・					

図 6.9 アクセス制御行列

あるが，はるかに経済的に組込む方法が種々存在する．1つの方法は，MULTICS など幾つかのシステムで採用されているもので，システム中の各ファイルごとにアクセス制御表（ACL）を対応させるものである．ACL には，このファイルのアクセスを認められた利用者と認められたアクセス権を一覧表にしている．プロジェクトごとのように利用者をグループに分け，“その他”という特別のグループを作ると ACL の長さを大幅に減少させることができる．

たとえば，DONOVAN が所有するファイル DATA を MADNICK および CARPENTER だけに読出しを認めようとするならば，ACL は次のようになる：

ファイル	アクセス制御表
DATA	DONOVAN (ALL ACCESS), MADNICK (READ), CARPENTER (READ), ALL OTHERS (NO ACCESS)

この方法では，4つの ACL 項目だけが必要であり，完全なアクセス制御行列のように5,000は必要としないことに注目してほしい．たいていの利用者はきわめて限定した範囲でのみ共有を認めるので（公共的なファイルは利用者の大集団に対して特に制限のないアクセスを認めて“その他”の欄を使用する），ACL は通常きわめて短い．ACL の情報は，BFD 項目の一部として能率よく

格納することができ，ファイルが使用中のときには AFT の項目の中に簡単に複写することができる．このようにすると，アクセス権の検査を効率よく行なうことができる．

6.6.2 合言葉

ACL およびこれに類する機能は大容量の記憶領域を必要とする．そのうえに，これらの表は可変長であるので，管理が複雑である．管理の機構としては，ACL 項目をすべて連鎖するか，あるいは，最大長の ACL を格納するだけのブロックを確保するかしなければならない．

アクセス制御を行なうための別の方法は**合言葉**を用いるものである．ファイル・ディレクトリの中の各ファイルに対して1つの合言葉が与えられる．このファイルのアクセスを要求する利用者は正しい合言葉を用意しなければならない．他の利用者にファイルのアクセスを許そうとするときには，そのファイルに対する合言葉を教えてやるだけでよい．この方法の利点は，情報の保護のために，各ファイルについてごくわずかの固定長領域しか必要でないという点にある．欠点は，保護のための情報はシステム中に格納されているので，不正なシステム・プログラマはすべての合言葉を得ることができることである．この欠点は，ACL の場合にも当てはまる．

合言葉の別の欠点は，アクセス制御の変更が容易でないことである．利用者が選択的に合言葉を訂正しようとするときどのようにすればよいか．合言葉を与えられている利用者の1人によるアクセスを拒否しようとするときどのようにしてこれを行なえばよいか．合言葉を変更することは可能であるが，そのときにはアクセスを許されている利用者すべてにそのことを通知しなければならない．

6.6.3 暗号法

合言葉を用いる方法も ACL を用いる方法も，共に，“アクセスの鍵”がシステム中に恒久的に格納されている点に欠点があった．他の方法は，すべてのファイルを暗号を用いて符号化することである．すべての利用者は符号化されたファイルをアクセスすることができるが，参照を許された利用者だけが内容の復号法を知っている．

ファイルの復号（読出し）と符号化（書込み）は，アクセス制御確認モジュールによって行なうことができる．要求を出す利用者は引数として符号キーを

用意する．利用者は必要なときその引数を変更することができる．情報の符号化には多数の方法が存在する（Skatrud, 1969）．1つの方法は，単純に，そのキーの値を用いて乱数発生を行なうことである．符号ルーチンは，符号化するファイルのバイトに次々と乱数を加算して行く．復号は，元のデータを得るために，この数を減じて行く．参照を許された利用者は，符号化で使用されたキーを復号のために使用して，乱数発生を開始する方法を知っている．

この方式では符号キーをシステムの中に格納しておく必要がない．利用者は，ファイルを符号化したり復号したりする間だけ，符号キーをシステムに入れればよい．したがって，不正直なシステム・プログラマが符号キーを読出してアクセスを可能とするような表は存在しない．暗号法は，キーをシステムの中に格納しないという利点を有するが，ファイルの符号化と復号に経費がかかる．

6.6.4 要　約

本節で述べたアクセス制御機構は，単独にでも，あるいは，組合せてでも使用することができる．個人の秘密，安全性，保護，共有，およびアクセス制御の問題は多数の研究者の注目をあびている．詳細については文献を参照してほしい：

Comber (1969), Conway (1972), Graham (1967), Graham and Denning (1972), Hoffman (1969), Lampson (1969, 1971), Molho (1970), Needham (1972), Scherf (1973), Schroeder (1969), Skatrud (1969), および Weissman (1969).

6.7 論理ファイル・システム

論理ファイル・システムは，論理レコードの構造体を物理ファイル・システムで用意される一次元的なバイト列としてのファイルに写像する．特に，レコードの要求をバイト列の要求に変換する．

第6.2節の例のファイル・システムでは，論理ファイル・システムは，レコード番号を基にして，任意の等長のレコードを直接アクセスする機能を与えた．一般のファイル・システムおよびデータ管理では，このほかの多くのレコード編成を用いることができる．この機能はしばしば**アクセス法**と呼ばれ，次の形式のレコード処理が行なわれる：

1. 順編成の固定長レコード（順アクセスおよび直接アクセス）
2. 順編成の可変長レコード（順アクセスおよび直接アクセス）
3. キーつき順編成レコード
4. 多重キーつきレコード
5. 鎖状編成レコード
6. 関係づきレコード——3項編成レコード

6.7.1 順編成固定長レコード

このレコード編成では，図 6.10 に示すように，ファイルには固定長レコード（レコードの長さはすべて等しい）が順番につながっている．この編成は“カード・イメージ”や“印刷イメージ”，スプール・ファイルなどの格納に向いている．

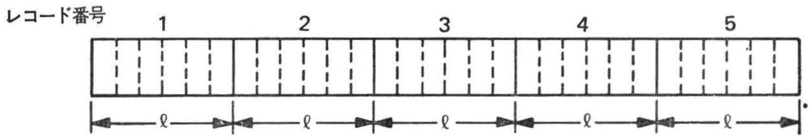


図 6.10 順編成固定長レコード（すべてのレコード長はℓバイト）

6.7.1.1 順アクセス

順アクセスに対しては，利用者は要求を出す度に“次の”レコードを求めている．これはカード読取り機からカードを読込むのと同じである．この要求はレコード番号が省略された形となる：

READ FILE(ETHEL) NEXT INTO LOCATION(BUFFER)

この機能を組込むには，論理ファイル・システムで現在の論理バイト番地(CLBA) をファイルの AFT 項目の中に格納しておかなければならない．ファイルが最初開かれるときには，論理バイト番地は 0 に設定される．したがって，各要求を処理した後で論理バイト番地は次のように更新される．ただし，RL はレコード長とする：

CLBA=CLBA+RL

6.7.1.2 直接アクセス

直接アクセスでは，利用者は必要とするレコードを指定する．この要求は前述の例と同様である：

READ FILE(ETHEL) RECORD(4) INTO
　　LOCATION(BUFFER)

論理バイト番地は簡単な式で計算される．ただし，RN はレコード番号とする．

$$\mathrm{CLBA}=(\mathrm{RN}-1)\times \mathrm{RL}$$

6.7.2 順編成可変長レコード

このレコード編成では，ファイルは不等長なレコードが順番につながっている．この編成は，図 6.11 に示すように，名前，職歴，印刷出力（後に続く空白を省略したもの）などを格納するのに向いている．

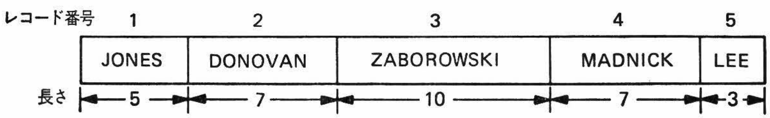
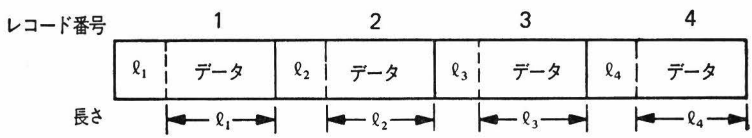


図 6.11 順編成可変長レコード

このようなレコードを格納する方法は種々存在する．この場合には，各レコードの境界を識別する必要があることに注意してほしい．たとえば，図 6.11 では1つのレコード JONESDONOVAN ではなく，2つの別個のレコード JONES および DONOVAN が存在することがどのようにしてわかるのであろうか．図 6.12 はこの問題を処理するのに利用可能な書式を示している．この方式では，各レコードの長さがファイルに格納される．



ℓ_n はレコード番号nのレコード長を示す

図 6.12 順編成可変長レコード

6.7.2.1 順アクセス

順アクセス要求は次の型式となる．ただし，変数Nは読込んだレコードの長さに設定される：

READ FILE(NAMES) NEXT INTO LOCATION(BUFFER)
LENGTH(N)

バッファ領域の長さは，ファイルの中の最も大きいレコードより小であってはならない．

図6.12の長さ指定 ℓ_n が2バイトであり，レコードは最大64Kバイトの長さまで可能であると仮定しよう．論理バイト番地がつねに次レコードの長さ指定の先頭に設定される(初期値は0である)と仮定すると，長さNは論理バイト番地が指す2バイトから求めることができる．次に，レコード自体はCLBA+2番地のNバイトとして読出すことができる．この2つの手順のためにバイト番地と長さを決定すればよいので，物理ファイル・システムを用いて，これらの要求の処理を行なうことができる．次の要求の準備として，論理バイト番地は次の式で更新される：

CLBA=CLBA+2+N

6.7.2.2 直接アクセス

直接アクセス要求は次の形となる：

READ FILE(NAMES) RECORD(3) INTO
LOCATION(BUFFER) LENGTH(N)

順編成可変長レコードを直接アクセスするための論理バイト番地は，基本的にはそのレコードを見つけ出すまで，レコードを順にたどって行って求めることができる．すなわち，レコード3が要求されたとすると，順番にレコード1，2をアクセスして求めた2つのレコードの長さを ℓ_1 および ℓ_2 として，論理バイト番地は次のように求められる：

$$CLBA=\ell_1+2+\ell_2+2$$

直接アクセスはきわめて効率の悪いものとなるおそれがあるが，次のようにして効率を改善することができる：

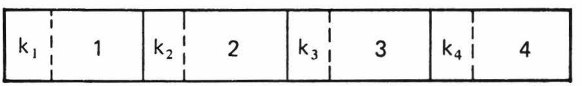
1. 最後に要求されたレコードの番号と番地を記録しておく．次の要求でレコード番号が前にアクセスしたより大きければ，先に進む．小さければ，ファイルの先頭から順番を数え始める．
2. レコード番号と論理バイト番地の対応表を作る．
3. レコードをアクセスする度に，次の使用のために，論理バイト番地を利用者プログラムで記録する（たとえば，IBM OS/360 の NOTE/POINT 機能）．このようにすると次回以降のアクセスがきわめて高速となる．

6.7.3 キーつき順編成レコード

別の種類のアクセスは，レコード番号や番地を用いずに，内容に基づくものである．たとえば，MADNICK の給与支払い情報を格納しているレコードの要求を考えてみよう．このような1つのキーを持つレコードは図 6.13 のように示すことができる．ただし，$k_1, \cdots, k_n$ はキーを示し，k_4=MADNICK としておく．

レコードはすべてキーの値が大きくなる順番で並べられている．たとえば，MADNICK は DONOVAN の後で，ZEIRING より前である．

順アクセスあるいは直接アクセスのための論理バイト番地は，正しいキーを



$k_1 < k_2 < k_3 < k_4$

図 6.13 順編成単一キーつきレコード

	キー	番地
k_1	ADLER	
k_2	DONOVAN	
k_3	MADNICK	
k_4	ZIERING	
	⋮	⋮

図 6.14 図 6.13 に対する指標表の例

持っているレコードを探すことによって求めることができる．このような手間のかかる探索を避けるために，別個の**記号表**や**指標表**ファイルが，レコードのキーと論理バイト番地の対応を示す目的で，使用される（図 6.14 参照）．

実際のシステムでは，指標表は幾つかの階層に分けられている．図 6.15 は，IBM の**指標つき順アクセス法**（ISAM）で採用されている構造を示している．主指標は 2 次指標を指す．2 次指標は，データ項目の部分集合の開始番地を格納している．

データ・ファイルのレコードの内 n 個おきのレコードは挿入のために空欄のままにしてある．要求を処理するには，主指標を 1 回アクセスし，2 次指標を 1 回アクセスし，部分集合のレコードを順番に探索する必要がある．

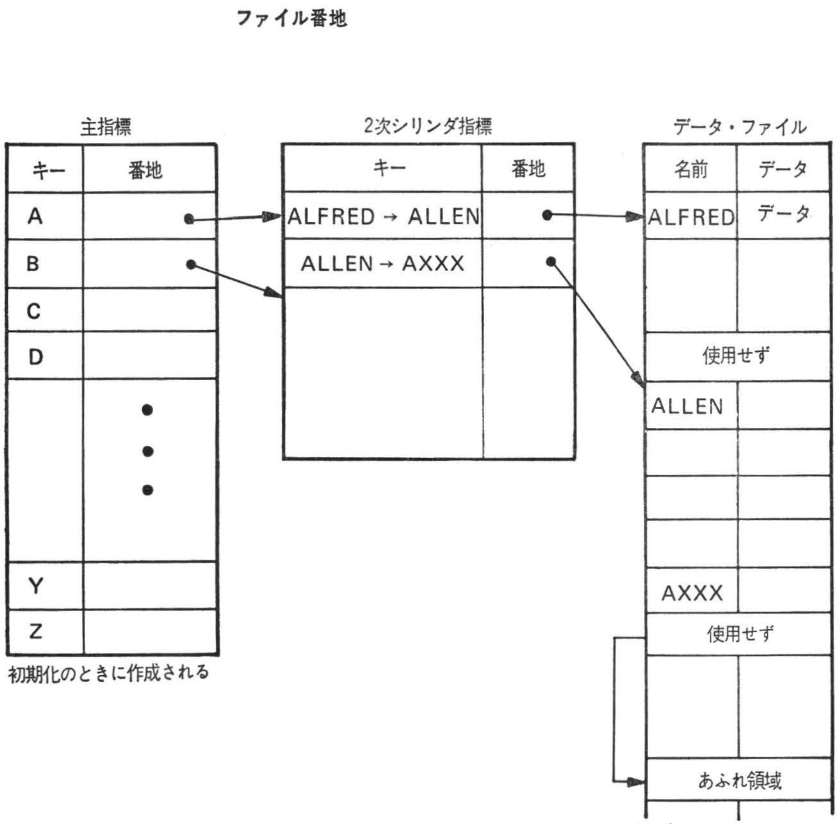


図 6.15 指標つき順編成

この方式は次の条件の下ではうまく動作する：

1. ほとんどの項目が最初にすべて一度に作成され，分類された形式で提出されること．
2. あふれ領域の使用を最小とするために，挿入や削除はごく一部分（10%以下）であること．
3. キーの順番で参照するような処理であること．

6.7.4 多重キーつきレコード（逆ファイル編成）

レコードの中の次の4つの部分はすべて同一の人を示す：

ID＝1743
NAME＝DONOVAN
SS#＝030－34－2674
JOB＝FASTEST PROFESSOR AT MIT

時として，利用者は，幾つかのキーの任意の1つで参照できるように，データを編成したいことがある．これは，図書館で本の検索を，書名，著者名，分類別によって行なうのと類似している．

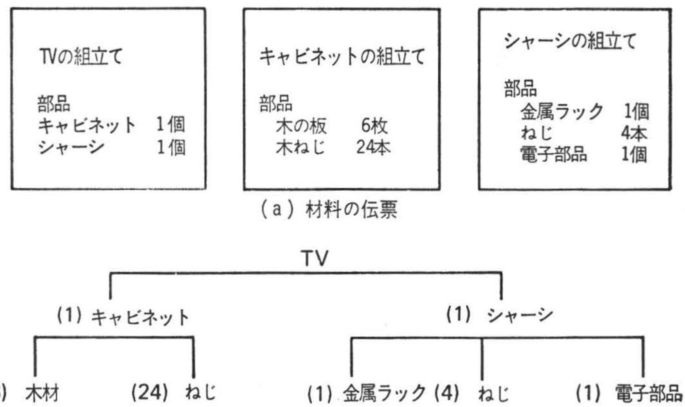
このための主要な技法は次のものである：

1. 多重の指標（指標表はデータの容量より大きくなるかもしれないことに注意）
2. 同一のキーを有するレコード同志を連鎖すること

6.7.5 鎖状編成のレコード

キーつきや多重キーつきレコードの編成は，普通には，基本ファイル・システムとしては提供されず，基本ファイル・システムの上に構成される．これらはデータ管理システムであり，複雑なレコード構造に対する処理機能を与える．鎖状編成は，より複雑なデータ管理機能を可能にする．

図 6.16 **a** には1つの例が示されている．この例は，1台のテレビジョン・セットを組立てるための“部品伝票”（部品表）である．図 6.16 **b** は同じ情報を“樹”あるいは“連鎖”の形で表わしている．このような構造を用いると，**上から下へ**，たとえば，1台のテレビにはネジが何本必要か，というアクセスや，**下から上へ**，たとえば，ネジを必要とするのは何の組立てでか，というアクセスが可能となる．このようなシステムの組込みの議論は，Chapin (1969)



(b) 製造の流れ

図 6.16 リスト構造のレコードの例

や Senko (1973) などの文献にゆずる.

6.7.6 関係づきレコード——3項編成レコード

基本ファイル・システムの上に構成することのできる別のデータベースの編成は**関係づきレコード** (relational record), あるいは**3項編成** (triple record) である (Codd, 1970). 論理レコードは別のレコードとの"関係"を持っている. たとえば, Frank は John の父であるなど. 図 6.17 は3つの項目の関係を示している. このようなシステムでは, 利用者は次のような質問を行なうことができる:

JAMES の FATHER は誰か (すなわち, ?——FATHER——JAMES)
すべての FATHER をあげよ (すなわち, ?——FATHER——?)

このようなシステムについては Codd (1970) および Senko (1973) が論じている.

レコード1	—	関係	—	レコード2
FRANK	—	FATHER	—	JOHN
JOHN	—	BROTHER	—	PAUL
JOHN	—	FATHER	—	JAMES

図 6.17 関係づきのデータベース

6.8 物理ファイル・システム

物理ファイル・システムの主たる機能は，論理バイト番地を物理ブロック番地に写像することである．図6.5の構成では，論理ファイル・システムは物理ファイル・システムを呼出して，要求した情報の論理バイト番地と長さとを引渡した．物理ファイル・システムは，まず割付け法モジュール（書込み要求のとき）を呼出し，次に装置駆動モジュールを呼出すか，あるいは直接，装置駆動モジュールを呼出すこともある．

物理ファイル・システムは，論理バイト番地から物理的な記憶域のブロックへの写像の情報を管理する．前節では，ファイルの物理的な記憶域の構成が単純であったので，この写像関数は単純であった（物理ブロック番地＝ファイルの先頭ブロック番地＋論理バイト番地/ブロック容量）．すべてのブロックは，図6.2のように，2次記憶装置に連続して格納された．

本節では，さらに次の3つの点を簡単に論じよう：

1. 入出力動作の最小化
2. 論理レコード長を物理ブロック容量に独立とすること
3. ファイル領域は不連続な割付けでもよいようにすること

6.8.1 入出動作の最小化

ファイル ETHEL（図6.2参照）のすべてのレコードを順番に読込むのに図6.4に示したアルゴリズムを用いて7つの論理レコードを読込むと，7回の入出力動作が必要である（各動作によって物理ブロック1つを読込む）．ファイルは全体でただ4つの物理ブロックを占有しているにすぎないから，余分な動作を除去して入出力動作の回数を減小させられないかと考えるであろう．

どの物理ブロックが主記憶に入っているかを記憶しておけば，入出力動作の回数を減少させることができる．どのブロックを主記憶に入れておくかを決定するには，きわめて複雑なアルゴリズムを用いることもできる．入力動作を行なう度に，物理ブロックが1つ主記憶中のバッファに複写される．現在バッファの中にどの物理ブロックが入っているかを記憶しておけば，図6.4の手順5を変更することによって，希望するブロックがすでに読込まれているか否かを検査することができる．そのブロックがすでにバッファの中に入っていれば，入力動作（手順6）を省略することができる．

ファイル ETHEL の場合には，レコード1を要求すると物理ブロック6がバッファの中へ複写される．その後でレコード2を要求すると，入出力動作を必要とせずに，バッファの操作で処理される．レコード3を要求すると，物理ブロックがバッファの中に複写される……このようにして，ファイル ETHEL は順番に処理すると，7回ではなく4回の入力動作で全部処理することができる．

この技法はしばしば**ファイル・バッファリング**と呼ばれ，多数の変形がある．たとえば，開かれたファイルごとに別々のバッファが割付けられる場合もあるし，ファイル全体でバッファを共用する場合もある．さらに，順アクセスに比べると影響の大きい乱アクセスの有害な作用を小さくし，入出力の並行動作を可能とするように，1つのファイルに複数のバッファを割付けることが望まれることもある（“ダブル・バッファリング”という言葉はここからきた）．効率のよいバッファリング法の問題は，第3.6.2節で論じたページ置換えアルゴリズムや第5.3.2節で論じた装置管理のバッファリングに関した問題ときわめてよく似ている．

6.8.2 論理レコード長を物理ブロック容量に独立とすること

第6.2節の簡単なシステムでは，1つの物理ブロックには整数個の論理レコードを収容し，余りの部分がないようにしなければならなかった．しかし困ったことに現実には，プログラマは物理ブロック長の約数でないような論理レコードを処理したいことがある．この問題を解決するには多くの方法がある．たとえば，13バイトの論理レコード50個を1000バイトのブロックに20バイトの論理レコードを作って格納することもできる．この場合，各論理レコードの13バイトは実際のレコードで，7バイトは無駄となっている．この方法は正しく働くが，効率よく2次記憶の領域を使用していない．

多くの記憶装置では，個々のトラックの書式にある程度の柔軟性を有している．たとえば，IBM 2314 や 3330 などの装置では，プログラマは1000バイトのブロック16個から500バイトのブロック30個までの間でトラック書式を変化させることができる（図 6.1 参照）．この書式変更は，ブロックの長さに拘らず各ブロックごとにレコード間隙のために一定領域が必要となるので，比例的な関係でない（第5章参照）．

一般に，トラック書式を変更するやり方はあまり具合のよいものではない．より一般的な手法は，図6.4の手順5および7を図 6.18 に示したように変更して，論理レコードが2つ以上の物理レコードにまたがってもよいようにする

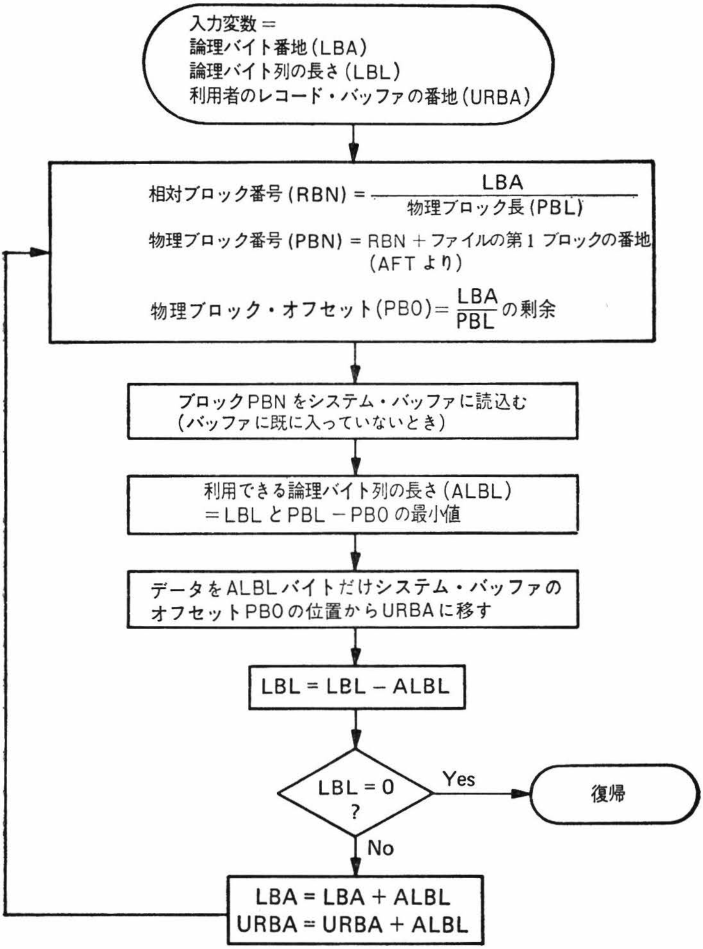


図 6.18 図 6.4 の段階5，6および7の改訂版

ことである．この手法では物理ブロックの最大長よりも大きい論理レコードの処理も可能である点が優れている．

6.8.3 不連続のファイル領域割付けを可能とすること

第6.2節の簡単なシステムで仮定したような，ファイル領域の連続的な割付

けは，必ずしも実際的ではないし，またつねに可能であるわけでもない．第6.9節（割付け法モジュール）には種々の理由が示されている．ファイルの物理ブロックが連続でないとしたら，論理バイト番地を物理ブロック番号に写像するには別のアルゴリズムが必要である．よく行なわれる2つの技法は次のものである：

1. 連鎖したブロック
2. ファイル・マップ

これらの基本的な技法には多数の変形も存在する．

6.8.3.1 連鎖したブロック

連鎖したブロックの写像では，各物理ブロックは“論理的に連続する”次ブロック番地を格納している．図6.19はブロック6，9，10および14を割付けられたファイルETHELを示している（図6.2と比較せよ）．基本ファイル・ディレクトリ項目は，以前と同様に，第1ブロックの番地を納めているが，それ以降のブロックの番地は番地ポインタ（しばしば，鎖と呼ばれる）を用いて求められる．

番地ポインタは，ブロックの容量を少し減少させて，(1000バイトでなく998

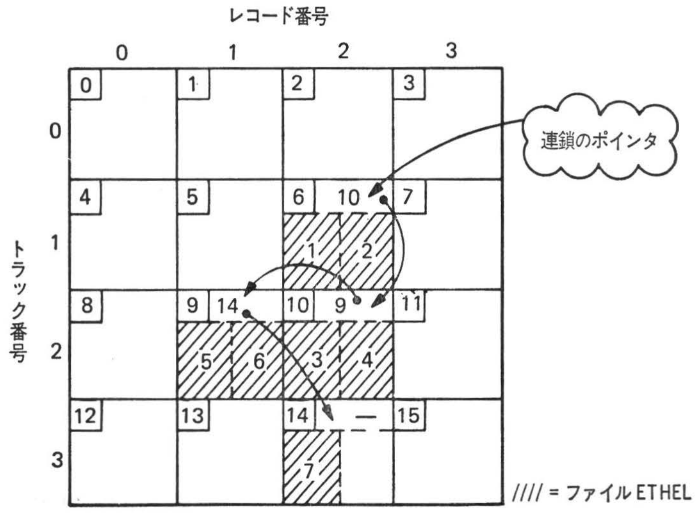


図 6.19 連鎖されたブロック

バイトとなる)，物理ブロックの中に格納されたり， IBM 3330 などの装置のように，データ・ブロックのヘッダ部（キー部）の一部として格納されたりする．

ブロックを連鎖する手法は，ブロックを順次アクセスする場合には能率がよいが（論理レコードの順アクセスに対応する)，第 6.7.2.2 節で論じたのと同様の理由によって，直接アクセスに対しては全く能率がわるい．連鎖ブロックの技法は第9章で述べる CTSS システムで採用されている．

6.8.3.2 ファイル・マップ

非連続的な領域割付けの別の方法は，各論理バイト番地を物理ブロック番地に写像するためのファイル・マップ表を使用するものである（第3.6.2.1節で論じたページングで使用したページ・マップと同様である)．このファイル・マップは，基本ファイル・ディレクトリの一部として，あるいは別個のブロックに格納される．図 6.20 は後者の方法を示している．ブロック6はファイル・マップを格納するのに用いられ，ブロック 7，9，10および 14はデータ・レコードを格納するのに用いられている．

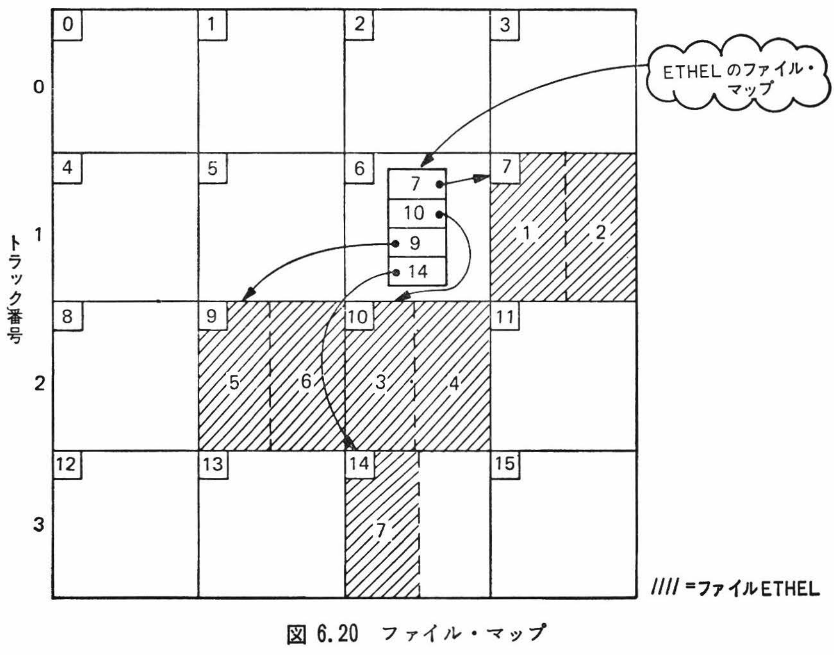


図 6.20 ファイル・マップ

ファイルが開かれるとき，ファイル・マップの全部，あるいは，ファイルが特に大きいときにはその一部がアクティヴ・ファイル表の情報の一部として主記憶に複写される．このようにすると写像を連続割付けと同様に能率よく行なうことができる．写像関数は次式のようになり，指標による表のアクセスと同様になる：

$$\text{物理ブロック番地}=\text{ファイル・マップ}\left(\frac{\text{論理バイト番地}}{\text{ブロック容量}}\right)$$

ファイル・マップがきわめて大きいときには幾つかの物理ブロックを必要とすることがある．このようなことは第9章で述べる IBM VM/360 システムにみられる．

6.9 割付け法モジュール

システムによっては，ファイル領域を割付けるのに，第6.3.1節で論じたように，CREATE 制御カードの LOCATION 部を用いて人手で行なわねばならないことがある．このためには，プログラマが領域の管理をしなければならず，通常10,000から100,000ブロック以上ある記憶装置を利用するときにはきわめてやっかいなことである．また，通常，多数のプログラマが同じ記憶装置を使用するので，プログラマが空き領域を求めるには，別の人のファイルをも自分で管理しておく必要がある．連続割付けを仮定すると，次の制御カードではファイル ETHEL と JOHN が重なってしまい，単純なファイル・システムでは誤りとしても検出されない．（このような重なりは特殊な場合には必要とされるが，普通には人手による割付けの誤りを意味している）：

CREATE ETHEL, RECSIZE=500, NUMRECS=7,
LOCATION=6
CREATE JOHN, RECSIZE=100, NUMRECS=30,
LOCATION=8

上述の理由によって，プログラマは単に次のように指定し，ファイル・システムが必要な領域を割付けるようにするのが望ましい：

CREATE ETHEL, RECSIZE=500, NUMRECS=7
CREATE JOHN, RECSIZE=100, NUMRECS=30

一般に，プログラマにとってファイルが格納される場所はどうでもよいことで

ある．

6.9.1 自動的割付け

システムが領域管理を行なう技法は多数存在する．1つの簡単な方法は，図6.3および図6.6bに示したように，割付けられていない一連のブロックごとに1つの“擬似”あるいは“空き”ファイルを作ることである．ETHELのための4ブロックの要求を満たすためには，ファイル・システムは4ブロック以上の領域を表わす空きファイルのディレクトリ項目を探す．図6.21は上述の制御カードの処理を示している．

たとえば，IBM OS/360は同様の自動割付け技法を用いている（詳細は第9章参照）．

欄番号	名前	論理レコード長	論理レコード数	第1物理ブロックの番地	物理ブロック
1	MARILYN	80	40	2	(2, 3, 4, 5)
2	空き	1000	4	6	(6, 7, 8, 9)
3	STU	132	5	10	(10)
4	空き	1000	5	11	(11, 12, 13, 14, 15)
5					
6					

(a) CREATEの処理前

欄番号	名前	論理レコード長	論理レコード数	第1物理ブロックの番地	物理ブロック
1	MARILYN	80	40	2	(2, 3, 4, 5)
2	ETHEL	500	7	6	(6, 7, 8, 9)
3	STU	132	5	10	(10)
4	JOHN	100	30	11	(11, 12, 13)
5	空き	1000	2	14	(14, 15)
6					

(b) CREATEの処理後

図6.21 自動的なファイル領域の割付け

6.9.2 動的割付け

上述の自動割付け法に関しては次の2つの問題点が存在する：

1. 2次記憶装置に断片を生じる．
2. しばしば，利用者はファイル容量がどうなるかあらかじめわからないことがある（コンパイラの出力オブジェクト・デックのファイルは入力原始プログラムの大きさと性質によって定まる）．

主記憶の断片化の解消の場合と同様に，2次記憶領域をより効率よく利用するには次の2つの技法が使用できる：

1. 再配置．たとえば，IBM 1130のディスク・モニタ・システムなどのシステムでは，定期的にディスク上のファイルの詰め合せを行なう．
2. 不連続なファイル領域の割付け．

不連続な領域の割付けを行なうと，断片化を防止できるばかりではなく，プログラム実行中にファイルの大きさを動的に増大させることも容易である．これらが，第6.8.3節で紹介したファイル・マップや連鎖したブロックを用いる動機となっている．

6.9.2.1 空きファイル・マップ

図 6.21 に示した割付け技法は，少数の容量の大きな空き領域が存在する場合にはうまく働く．小容量の空き領域が多数存在する場合には，ファイル・ディレクトリがきわめて大きくなり，効率が悪い．この問題の1つの解決法は，すべての空きブロックの表を収容する1つの特別なファイル・マップを使用することである（第 6.8.3.2 節のものと同様）．1つのブロックを割付ける必要があるときには，そのブロックはその表から取除かれる．1つ以上のブロックが解放されたときには（ファイルが抹消されたとき），これらのブロックは空きファイル・マップに追加される．

6.9.2.2 空きブロックの鎖

別の割付け法は，すべての空きブロックを連鎖するものである（第 6.8.3.1 節と同様の方法）．この場合には探索が不要である．ブロックが必要なときには，鎖の最初のものを取出す．逆に，ブロックが解放されると，このブロックは鎖の始めにつながる．

6.9.2.3 ビット・マップ

別のよく用いられる手法は，**ビット・マップ**と呼ばれる一連のビット列を用いて各ブロックの状態を示すものである．図 6.21 a の16個のブロックは，16ビットの列を用いて次のように表わすことができる．ただし，1は対応するブロックが使用中，0はそのブロックが空いていることを示している：

1111110000100000

この技法は第9章で述べる IBM VS/370 で用いられている．

ビット・マップ方式の主たる利点は，マップがきわめて小さいので，主記憶中で管理できるということである．したがって，割付けを高速で行なうことができる．

6.10 装置駆動モジュール，入出力起動モジュール，装置ハンドラ

これらのルーチンの基本機能は次のとおりである：

1. 物理番地を装置番地に写像する（たとえば，ファイル ETHEL のレコード4の物理番地は物理ブロック7で，装置番地はトラック1，レコード3である）．
2. チャネル・プログラムを生成する．たとえば，このプログラムは次のことを行なう．
   a．トラック1をシークする．
   b．トラック1の中を探してレコード3を求める．
   c．ディスクが回ってくるのを待つ．
   d．読出す．
3. 入出力を要求する．すなわち，入出力スケジューラを呼出してこの入出力をスケジュールさせる．
4. 入出力割込みおよび発生する誤りの処理を行なう．

これらのルーチンのアルゴリズムはすでに第5章（装置管理）で示されている．

6.11 情報管理の動向

ファイル・システムと第3章で論じたセグメンテーションおよびページングの原理との類似点に注目してみよう：

1. ファイルとセグメントの概念は，情報の論理的な編成に関係している（すなわち，ファイルはセグメントに対応する）.
2. ファイルとセグメントも任意の容量で，動的に容量が大きくなり得る.
3. ファイルとセグメントは2次元番地指定が必要（すなわち，ファイル名とレコード番号はセグメント名とバイト番号に対応する）.
4. ファイル・ディレクトリは機能的にセグメント・マップに対応する.
5. ファイル・マップは機能的にページ・マップに対応する.
6. 主記憶のバッファは機能的にページを収容する主記憶ブロックに対応する.

これらの類似点を認識したうえで，組込みには次の2とおりの方法がある：

1. 類似性を無視して，ファイル・システムとページング／セグメンテーションの機構を別個に構成することができる（たとえば，IBM の OS/VS 2 と VM/370).
2. ファイル・システムとセグメンテーション／ページング機構とを一体化することができる.

記憶装置と情報管理とを一体化したシステムの設計では，そのためのハードウェアを設計することも可能である．たとえば，ファイルの読出しと書込みの多くの部分は，最初のアクセス（OPEN, CLOSE）だけをソフトウェアで処理することにして，それ以外はハードウェアにて行なうこともできる．図 6.22 はこの様子を示したもので，左側に MULTICS のセグメンテーションとデマンド・ページングの機構が示され，右側に対応するファイル・システムの機構が示されている.

図 6.22 のハードウェアは，外部セグメントに対する記号参照によって起動される．ハードウェアは，セグメントにセグメント番号が割付けられているかどうかを検査する（リンケージ・ビットが設定されている)．もしセグメント番号が割付けられていない場合には，割込みを発生して，要求されたセグメン

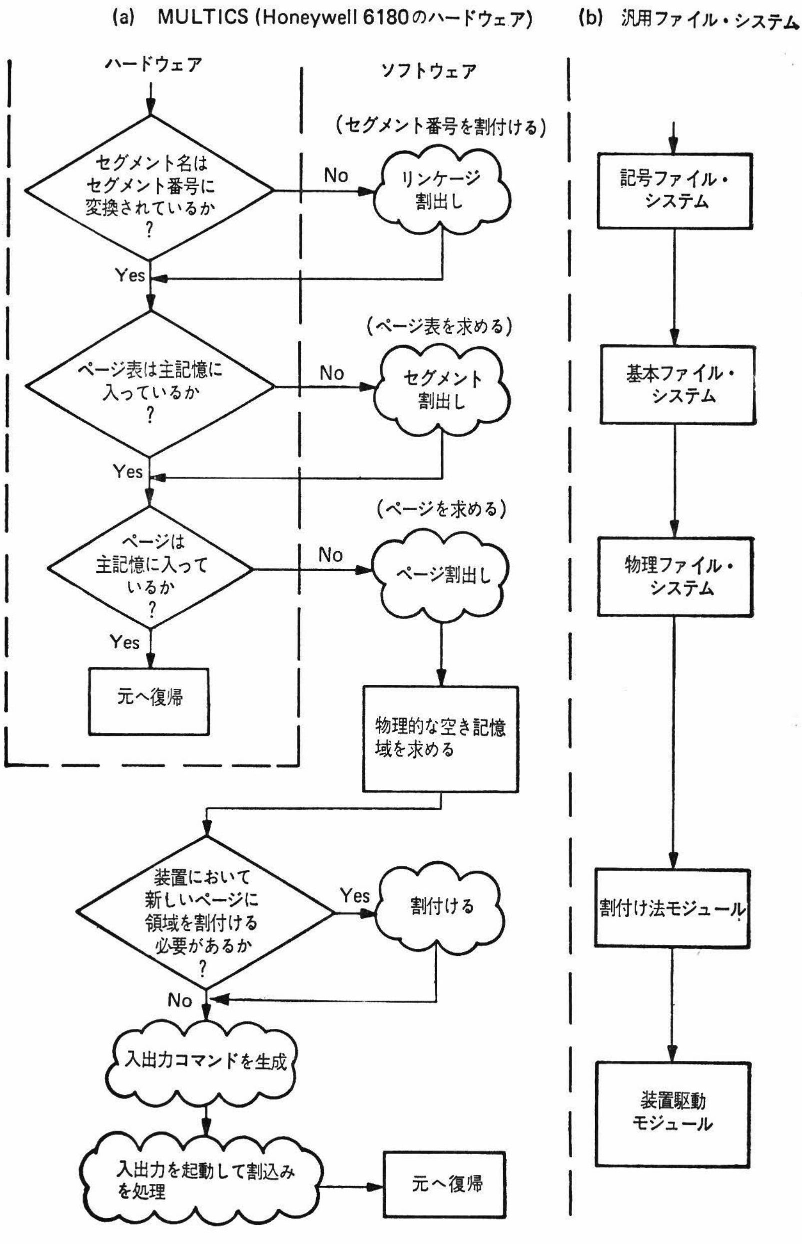


図 6.22 MULTICS ファイル・システムと汎用ファイル・システムとの比較

トをファイル・システムの階層の中で探索するソフトウェアのモジュールを起動させる．このハードウェアとソフトウェアのモジュールは記号ファイル・システムの機能を遂行する．すなわち，要求されたファイルを一意的に識別する．

ファイルが識別されると，ハードウェアはそのファイルのアクセスを完了させようとする．もし，そのファイルのページ表が主記憶中に入っていないならば，ハードウェアは割込みを発生させ，セグメント割出しのソフトウェアが起動させる．そこでソフトウェアはページ表を探してそれを主記憶の中にロードする．この機能はファイルを開き，アクティヴ・ファイル表を設定すること――基本ファイル・システムの仕事――に相当する．

そして再びハードウェアはファイルのアクセスを試みる．アクセスしようとするページが主記憶中に入っていないときには，ハードウェアは割込みを発生し，ページ割出し機構が起動される．この機能は論理的な参照を物理番地に変換する物理ファイル・システムの機能に相当する．

要求されたページをロードするために主記憶の領域が必要となると，ページ置換えアルゴリズムが呼出される（割付け法モジュールに対応する）．最後に，ページを主記憶へ移す入出力が遂行される．ページが主記憶に入ると，ソフトウェアとハードウェアはアクセスを完了させる．

将来は，記憶管理と情報管理の多くのモジュールを合併したシステムが多数構成されるであろう．

演　習　問　題

6.1

1. ファイル・システムにおいて OPEN および CLOSE 命令が用いられる目的は何か．
2. ファイル・システムからこれらの命令を削除したとして，この目的を果たすことは可能か．それにはどのようにすればよいか．
3. 問 2 の解答に対して，どんな欠点や非能率な点があるか．
（すなわち，普通には OPEN および CLOSE 命令を用いる理由は何か）．

6.2 本章で示したファイル・システムの設計では，入出力動作の実行回数を最小とするためにバッファを用いると有利であった．n 個のファイルが開かれており（使用中），n 個のバッファが存在するような場合を考えてみよう．（n＝3 と仮定し，ファイル名をＡ，Ｂ，およびＣとしておく.）

ファイルにバッファを割付けるには幾つかの方式が存在する．2つの可能性は次のとおりである：

- ファイルごとに特定のバッファを恒久的に割当てる．
- バッファを群として管理し，必要とされるときバッファを動的に割当てる（デマンド・ページング・システムによる記憶割付けと同様）．

1. 1つのファイルに2つ以上のバッファを割当てることの利点は何か．この方式の動作を説明するための要求列の例をあげよ．
2. ファイルにバッファを動的に割当てるための適当なアルゴリズムを述べよ．その理由を簡単に述べよ．（特に，いつバッファがファイルに割当てられ，どのバッファが選択されるか．）
   a．各ファイルに1つのバッファを用いる方式において，きわめて能率よく動作するファイル・システム要求の順番を述べよ．
   b．同様に，きわめて能率の悪い順番を述べよ（たとえば，バッファは役に立たず，毎回入出力動作が必要となる）．
   c．動的なバッファ割当て方式において，きわめて能率よく動作するファイル・システム要求の順番を述べよ．この順番が問bのものと等しければすばらしいことである．
   d．同様に，きわめて能率の悪い動作を行なう順番を示せ．この順番が問aのものと同一であればすばらしいことである．
3. 上の問cおよびdからして，どの方式を選択すればよいか．その理由は何か，判定の基準と仮定した点とを説明せよ．
4. バッファ割付けの問題に役立つような情報をファイル・システムに用意することは可能か．

6.3 ファイル全体に1つのファイル・ディレクトリが存在する場合に，ボリュームごとにファイル・ディレクトリを用いることは依然として有用か．その理由は何か．

6.4 図6.4で行なったようにして，次の READ 要求を満たす手順を示せ：

1. READ FILE(ETHEL) RECORD(1) INTO LOCATION(12000).
2. READ FILE(ETHEL) RECORD(2) INTO LOCATION(12000).

6.5 ファイル・システムの階層のどのモジュールが次の問題を処理するか，また，いかに処理するかについて述べよ．

1. 2次記憶域の断片化．

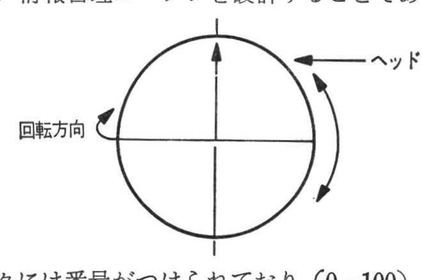
2. 別のファイルに同一の名前を許すこと.
3. 新しい論理構造を可能とすること. たとえば, キーつきファイル.
4. 新しい物理的な構成法を可能とすること.
5. バッファリング.
6. ブロッキング.

6.6 割付け法モジュールの節（第6.9節）では, 情報管理における記憶域割付けの技法を幾つか論じた（たとえば, 利用者による割付け, 自動的連続割付け, および動的割付け）. 第 6.9.1 節の自動的割付け方式は分割記憶割付け技法にきわめてよく似ている.

1. 記憶装置の領域の断片化は情報を詰め合せることによって解消できる. 普通のディスクやドラム装置は再装置レジスタを有していないが, どのようにすればファイルを再配置することができるか.
2. ファイルで詰め合せや再配置を行なうことは, しばしば避けられるが, その理由は何か.

6.7 任意の長さのファイル名を用いることができ, さらに, 利用者は名前づけに関して標準的な規約に従うものとするとき（たとえば, ファイル名に自分の名前を含めることなど）, 1つの層で1つの長い名前を持つファイル・システムと階層的なファイル・ディレクトリ・システムの間に相違点が存在するか.

6.8 読者の上司は幸運にも食料品店で700万ドルの価値のあるスタンプを抽き当てたので, 計算センターのために最新式の周辺装置である“球形ドラム”を購入した. このドラムは一方向に回転し, 各トラックの記憶容量は異なるが, 円状のトラックを有している. 読者の仕事はこの“神の創造物”を取扱う装置／情報管理ルーチンを設計することである.



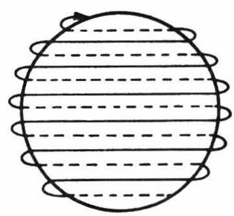
1. 各トラックには番号がつけられており（0—100）, 両端のトラック（0 および 100）の容量は中央のトラック（49, 50, 51）の容量の半分であ

る．また，ヘッドがトラックからトラックへ移動するのに10ミリ秒かかる．ボリュームのファイル・ディレクトリを格納するのに適した場所（トラック）はどこか．その理由は何か．

2. ドラムの違う場所にファイルを入れるとき料金を違えるのがよいか．料金の差が考慮される理由は何か．

3. この装置の割付けをどのようにすればよいか．（註：装置全体で単一のブロック長を用いるのは適当でないと仮定する．）

4. 読者は100万ドルを相続したので，球の周囲を螺施状に回るただ1つのトラックができるような機構に変換しようと決めた（次図を見よ）．ほかに購入しなければならないものや，問題点があるか．



5. この装置は何色に塗ればよいか．

6.9

1. 本章で論じた階層的ファイル・システムが与えられているとする．次に示す要素を取除くとどのような影響があるか．この削除の影響を補償するためにはどのような手段が必要とされるか．（各場合には，除外される要素以外のものは依然として存在しているものとする．）

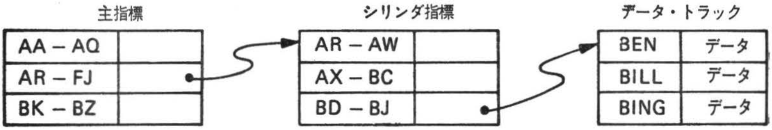
 a．ファイル・マップ

 b．プログラマの個人用ディレクトリ

 c．ファイル・ディレクトリの中の長さ指定部

2. ファイル・ディレクトリが自分自身のディレクトリ項目を格納している理由は何か．

6.10 第6.7.3節では，次に読出したり書込んだりするレコードを次のいずれかによって指定することができるような指標つき順アクセス法について述べた．



1. 次レコード（キーの順番に並べられている）
2. 特定のキーの値を持つレコード

指標つき順アクセス法では，ファイルが作られるときに，レコードをキーの順に並べて書込むことによって，1のアクセスを行なっている（すなわち，入力データはあらかじめ分類しておく必要がある）．2のアクセスは，3回の入出力動作によって行なう：

- 主指標（普通200以下の項目）を読出して探索し，データがどのシリンダに入っているかを決定する．
- シリンダ指標（普通20以下の項目）を読出して探索し，データがどのトラックに入っているかを決定する．
- トラック（普通40以下の項目）を読出してキーが一致するレコードを探す．

（註：200×20×40=160,000データ・レコードはきわめて大きい．）

1. 指標つき順アクセス型のアクセス法の別の組込みを考えよ．各種の表の探索法（たとえば，2分探索法，ハッシュ法），連鎖したレコードなどを考えてみよ．
（註：この問に答える前に他の問を読むこと．）
2. 次の順番のレコードを得るためには，指標つき順編成は1回か0回（バッファに入っている）の入出力動作しか必要としない（あふれレコードなどの場合を無視する）．問1で答えた方法は何回の入出力動作が必要か．
3. 指標つき順アクセス法でキーによりアクセスするには，ただ3回の入出力動作でよい．問1で答えた方法では何回の入出力動作が必要か．
4. 指標つき順アクセス法の場合に，バッファ領域を増すと入出力動作の回数はどのように減少するか．
5. 指標つき順アクセス法ではレコードの挿入や削除はうまくできない．挿入や削除を行なうと，データ・トラックが部分的に分類されていない状態になる（たとえば，BEN, BING, BILL などのようになって，順番

に読出すのが多少やっかいとなる). さらに, あまりにも多くのレコードが挿入されると新しいデータを“あふれ領域”に入れる必要が生じる. このようなことは, 順アクセスおよび直接アクセスの両方の場合に性能を低下させる. 問1の答えでは, 挿入や削除はどのように処理されているか. 指標つき順編成と比較して性能を述べよ.

6. 問1の答えと指標つき順アクセス法とを比較して, 利点と欠点とを簡単に述べよ. とくに, 問2および5で触れることができなかった点を述べよ.

6.11 IBM 社 (Interstellar Biodegradable Manufacturing Company) はレーザ・ホログラフィの記憶装置を開発した. この装置は主記憶装置と同程度の乱アクセス時間を有し (たとえば5**マイクロ秒**), ディスク装置に匹適する容量と経費である.

この装置は次の2つの奇珍な特性を有している：

- すべての情報は128バイトを単位として格納したり読出したりする必要がある (すなわち, 各番地は128バイトを収容する). 128バイトを10マイクロ秒以内で読み／書きすることが可能である.
- 書込みには高輝度のレーザを必要とするので, 1つのレコードは100回以上書換えることができない. それ以上では材質が劣化する. レコードは無制限の回数で読出すことができる.

この装置はファイル・システムの情報を格納するために使用される予定である. 現在のファイル・システムを変更してみよう.

1. この装置の速度を考えるとき, 複数の主記憶バッファを持つことは価値があるか. その理由を述べよ.

2. 書込み回数の制限を情報管理ではいかに処理したらよいか. 簡単明瞭にその方法を述べよ.

3. 問2の答えによって影響を受けるファイル・システムのモジュールは何か. 簡単に説明せよ.

6.12 多層のディレクトリを持つファイル・システムにおいて, 利用者がファイルを参照するのに完全に修飾された名前を使用するのはきわめて不便である. SYSCATLG. SYSMASTER. SYSDIR. GROUP. USER. ALPHA といったファイル名を完全な形で書かなければならないというのではな

く，プログラマがこのファイルを ALPHA という名前で呼べるようにしたい．これを可能にするためには，ファイル・システムはどれだけ余分の管理を行なわなければならないか．

6.13 第6.6.1節には，各ファイルに対するアクセス制御表（ACL）の原理が紹介された．ACL は，そのファイルの参照を認められたすべての利用者を示している．別の組込み法として，各利用者ごとに利用者制御表(UCL)が用意されている場合を考えてみよう．UCL は，その利用者に対して参照が認められているすべてのファイルを示している．UCL を用いる方法が ACL を用いる方法より優れている点があるか．どの方法をとればよいか．その理由は何か．

6.14 図 6.11 に示したような可変長レコードのファイルが編成されていると仮定して，直接アクセス読出しに対して使用することができる論理ファイル・システムのルーチンの流れ図あるいは PL/I プログラムを示せ（第6.3.3 節のものと同様な形式とせよ）．必要とされる仮定を述べよ．

（註：このルーチンは物理ファイル・システムのルーチンだけを呼出す.）

6.15 2次記憶領域の要求は，ファイル内でデータ圧縮の技法を用いることによって減少させることができる．1つの方式は，内容がすべて 0 のブロックには記憶域を割付けないものである．さらに，現在割付けられているブロックが変更されて，すべて 0 となったときには，領域を解放する．

1. この技法が大幅に領域を節約できるような場合を示せ．
2. この機能を組込むには，ファイル・システムのどのモジュールを変更しなければならないか．その理由をも述べよ．

6.16 多くのファイルではレコードが順番に，あるいは部分的に順番に使用されるので，記憶装置においてすべてのブロックをかためておくことが好ましい（すなわち，同一あるいは隣接のシリンダにおく）．動的割付けでブロックをかためるようなアルゴリズムを示せ（第 6.9.2 参照）．

6.17 階層的なファイル・ディレクトリを用いる場合には（図 6.8 参照），1つのファイルを一意的に識別するためには，完全な経路を指定する必要がある（たとえば，DONOVAN の SINE と MADNICK の MATH の SINE とは別の物かもしれない）．利用者プログラムでは完全な経路名を

指定することも可能であるし，簡単のために局所的な名前だけ（たとえばSINE）を指定してもよい．あいまいさを解決するために，どのディレクトリをどの順番で探索すればよいかを決定する探索則（無指定用と利用者指定用）が使用される．

1. 探索則と局所名とを利用する方法が，完全な経路名を使用する方法より優れている点は何か．
2. この方式で影響を受けるファイル・システムのモジュールは何か．
3. この探索則方式の組込みを示せ．追加が必要とされる制御カード，表，特殊ルーチンについて述べよ．

第 7 章

例題オペレーティング・システムの設計

7.1 は じ め に

本章では，オペレーティング・システムの原理と各種のモジュールの結合方法を理解する助けとなるようにとの目的で例題のオペレーティング・システムを示そう．

IBM の OS/360 や MULTICS など教育が主目的でないオペレーティング・システムと比べると，このシステムは単純で理解が容易である．また，このシステムはアセンブラ語にしてカード 2,500 枚以内の比較的小規模なシステムである．

このオペレーティング・システムには言語処理プログラムやユーティリティ・プログラムを含んでいない．しかし，このオペレーティング・システムには上層のスーパバイザやそれに関連したプログラムが容易に組込めるような，多重処理や動的記憶域割付け，および装置管理などの機能を与えるシステムの基本的な核が組込まれている．これらの機能は，オペレーティング・システムの基礎を学ぶために最も基本的なものである．

このシステムは，ジョブ流を処理するための簡単な上層スーパバイザを含んでいる．したがって，このオペレーティング・システムは，多数のオンライン装置を制御するなどの実際の応用にも使用可能であるという意味で，完備した形となっている．

最後に，このシステムは意図的にモジュール化および構造化を行なって設計

この例題のオペレーティング・システムは，John D. DeTreville の修士論文“例題オペレーティング・システムの設計と組込み”（M.I.T., 1972 年 6 月）に基づいている．Richard Swift や Paul Fredette などがデバッグを手伝った．この研究は，海軍研究契約第 4102 (01) により，国防省の上級研究計画局（ARPA）が研究費を支出し，M.I.T. の MAC 計画の研究の一部として行なわれたものである．

している．この結果，特に，処理装置管理，記憶管理，および装置管理に関する部分は容易に区別することが可能である．このシステムは IBM 360 を用いて組込んでいるが，そのアルゴリズムや技法はたいていの現代の計算機にも適応可能である．すべてのデータベースは，読解を容易とするように，PL/I 型の構造の宣言によって記述されている．詳細な組込みを学んだり実際にこのオペレーティング・システムを実行しようとする読者のために，IBM 360 アセンブラ語のプログラムも示してある．

7.2 システム概観

例題のオペレーティング・システムは，IBM 360 のために設計された多重プログラミング・システムである．任意の種類の入出力装置を用いることが可能であるが，現在のところは，カード読取り機および印刷機を取扱う機能だけを組込んでいる．しかし，利用者プログラムで非標準的な装置のためのルーチンを用意することが可能である．ファイル・システムは用意されていない．

利用者プログラムに対する記憶域の割付けは動的な区画割付けによって行なわれる．利用者は，ジョブに対して必要な記憶容量を，2K バイト単位で指定しなければならない．このシステムは現在のところ，自分のルーチンのために約 6K バイト必要で，残りの記憶域は利用者プログラムとシステムの管理表のために使用できる．

このシステムは，別々の入力装置（この場合は読取り機）から入力され，別々の出力装置（この場合は印刷機）に出力される複数のジョブ流を処理することが可能である．各ジョブ流には一群のジョブが図 7.1 に示したように順番に積まれている．

1つのジョブは，$JOB カードで始まり，1つの目的デックと任意のデータから構成されている．$JOB カードの書式は次のとおりである：

$JOB，記憶容量，名前＝装置種別，名前＝装置種別，…

その例を次に示す：

$JOB, 8K, FILEA=IN, FILEB=OUT

記憶容量はジョブで必要となる記憶容量を示す．上の例では 8K である．“名前＝装置種別”は必要な回数繰返してもよい．必要のないときは省略してもよい．“名前”は，利用者プログラムがこの装置を参照する名前を示す．こ

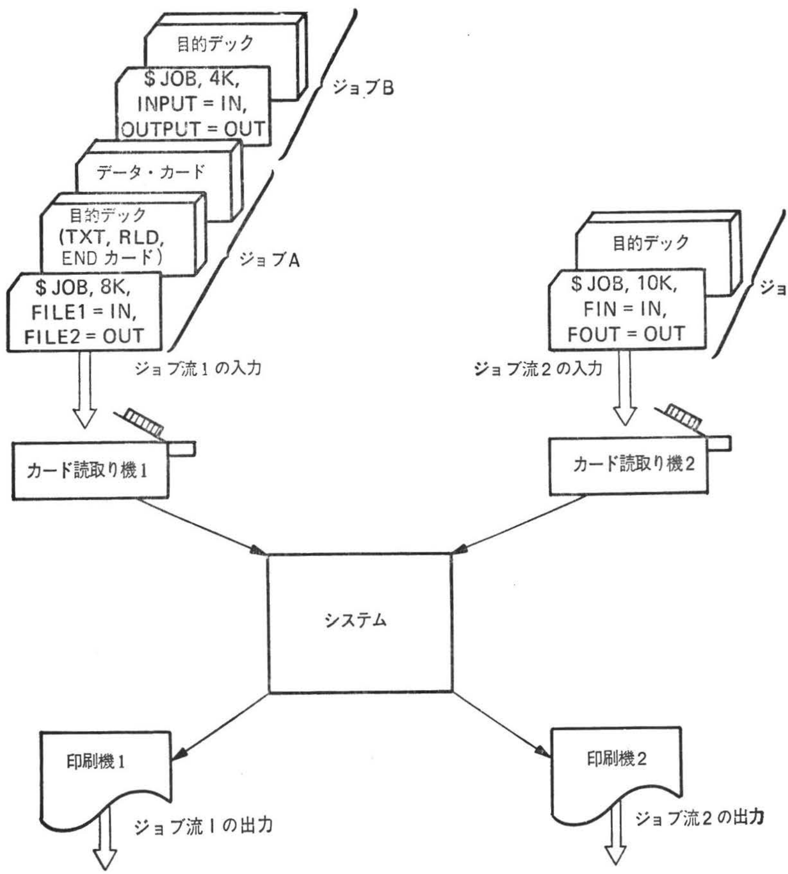


図 7.1 多重ジョブ流の多重プログラミング・システム概観

の名前は利用者によってつけられるものであり，柔軟で装置に独立な参照が可能となる．“装置種別”は割当てるべき装置の種類を示す．現在のところ，3種類の装置を使用することができる．

“IN”はシステム入力の装置を示し，このジョブ流に対するカード読取り機である．“OUT”はシステム出力の装置を示し，印刷機である．“EXCP”は，

利用者が自分のハンドラ・ルーチンを作らなければならないような非標準的な装置を示す．現在のところ，$JOB カードは，最大限，1つの IN と1つの OUT と1つの EXCP を参照する部分を持つことが可能である．

$JOB カードの直後に続く目的デックは，IBM OS/360 の標準の目的デックと同一の書式をとる．しかし，外部参照は許されない（すなわち，サブルーチン結合の機能は用意されていない）．$JOB カードに IN の指定が行なわれている場合には，目的カードに続いて利用者プログラムに対するカード入力データが置かれる．

利用者プログラムは並列処理を行なうことができ（すなわち，複数のプロセスを作ることができる），プログラムの並列実行部分の間で通信するための機能を用いて，並列処理を制御する機能がある．システム中の各種のジョブやプロセスを自動的にスケジュールする．もちろんスケジューリングは組込みを簡単にするために，限定されたものとなっている．このシステムには，もっと高度の機能を実現するために拡張することが容易である．

7.3 設計の概観

例題のオペレーティング・システムの設計は 図 7.2 に示した構成に従っている．

このシステムでは，プロセスという考えに基づいてシステムの概念を組立てている．プロセスを保持するのに必要な条件が幾つか存在する．適当な条件の下で，プロセスは幾つかの基本機能を呼出すことができる．不幸にして，今日の計算機のハードウエアでは，これらの基本機能が用意されていない．

したがって，まず第1に行なわなければならないことは，プロセスを保持するための基本機能（拡張計算機）を築くことである．これらがオペレーティング・システムの中心あるいは核となる．これらの基本機能の例はPおよびV動作，基本多重処理機能，およびトラフィック制御である．読者は，これらのソフトウェア機能がハードウェアの命令と同様にして実行されていると考えることができる．

7.3.1 拡張計算機の手法

核部分については，一組の拡張命令を備えた拡張計算機であると考えるのが最もよい．本章の組込みでは，この拡張命令はスーパバイザ呼出し命令を用いて作られている．たとえば次の命令列は記憶管理に対して，2語境界から始ま

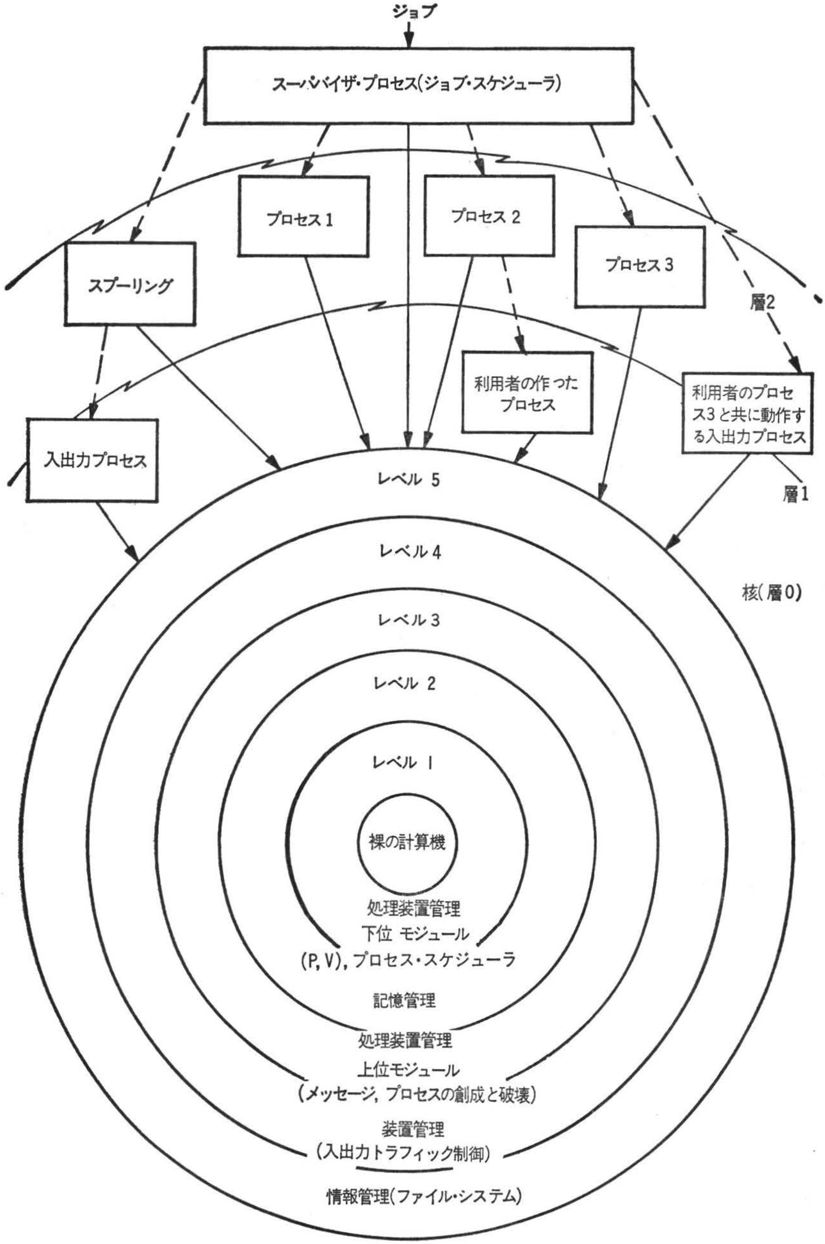


図7.2 オペレーティング・システムの階層構造

る32バイトの領域を要求する：

```
         LA    2,REQUEST     引数を設定する
         SVC   C'A'          記憶域を割付ける
         .
         .
         .
REQUEST  DC    F'32'         32バイト要求
LOCATION DS    A             システムによって指定される番地
ALIGN    DC    F'8'          2語境界へ調整
```

スーパバイザ呼出しの基本機能とそれらの呼出し手順については第7.5.2節に一覧表を示してある.

オペレーティング・システムのある種の機能は，システムの基本機能としてではなく，特別のシステム・プロセスによって実現できる．この例題システムでは，スーパバイザ・プロセス（ジョブ流のハンドラ）および装置ハンドラ・プロセスを含めて，幾つかこのようなプロセスが存在している．各ジョブ流は別個のジョブ流処理プロセスを持っており，使用する各装置ごとに別個の装置ハンドラ・プロセスが存在する．さらに，利用者のプロセスは別のプロセスを作ることができる．同一のジョブに属するプロセスは**プロセス群**と呼ばれる.

7.3.2　使用した概念の要約

このオペレーティング・システムの特徴の多くは，元々別のシステムで用いられていたものである．階層的なシステム組込みの概念は Dijkstra (1968) によって導入されたものであり，Dijkstra はプロセス同期のセマフォアの考えをも導入した．Madnick (1969) はファイル・システムの設計に階層的な手法を拡張した．プロセス間通信のためのメッセージ・システムの形式は Hansen (1970) によるものであり，Hansen は同時にシステムの核部分の概念を明瞭にした．プロセス制御に関する用語と原理の多くは Saltzer (1966) によって導入され，MULTICS に組込まれた．この分野のアイデアの幾つかは MULTICS から取られている．オペレーティング・システムは資源管理モジュールの集まりであるという考え方（および分割のし方）は Donovan (1972) によっている．加えて，幾つかの中心的な特徴は OS/360, MFT (IBM, 8), MVT (IBM,

1)，および CTSS (Corbató, 1972；Crisman, 1965) などのオペレーティング・システムから取られている．

7.4 例題オペレーティング・システムの段階と層

核部分の階層的な構成においては，底辺から上部への順で，各段階はそれより下の段階の存在にのみ関係し，それ以上の段階には関係しない．この手法は理解が容易で，デバッグがしやすく，新しい理論を発展させるのに適しているという利点がある．

例題オペレーティング・システムの5つの段階と層（あるいはモジュール）は次のとおりである：

段階
- プロセス管理，下位モジュール（最下位）
- 記憶管理モジュール
- プロセス管理，上位モジュール

層
- 装置管理
- スーパバイザ・プロセスのモジュール（最上位）

プロセス管理の機能は，記憶管理をはさんで両側に，下位のモジュールと上位のモジュールに分割される．その理由は，プロセス管理の一部分（上位モジュールの部分）は記憶管理ルーチンに依存し，記憶管理自体は，記憶管理の下に位置する必要のあるプロセス管理ルーチンの部分に依存しているからである．

スーパバイザ・プロセスのモジュールと利用者プログラムは，きわめてよく似た条件の下で動作している．このようなことは，拡張計算機の手法により可能となったものであり，オペレーティング・システムを単純にする．

7.4.1 プロセス管理，下位モジュール

プロセス管理の下位モジュールは，プロセス間の同期のための基本動作と共に，処理装置の多重化を行なう．このモジュールは，“多重プログラミングを保持する”モジュールであるとみなすことができる．

このモジュールは，実行適格のプロセスをスケジュールして実行させる．システム内のプロセスは，入出力動作の完了や他のプロセスからの信号などの外部の事象を待合せているのでなければ，通常，実行適格である．プロセスは，トラフィック制御によって単純なラウンド・ロビン方式でスケジュールされる．プロセスは，上に述べた理由で一時的に実行不適格となったり，割当て時

間と呼ばれる一定の時間――このシステムでは30ミリ秒――が経過するまで実行することができる．割当て時間が経過するとプロセスは停止されるが，後で実行を継続するために適格のままで留まる．いずれの場合にも，トラフィック制御は別のプロセスを選択して実行を開始させる．

このモジュールは，プロセスの同期のための基本機能をも提供する．この基本機能は**P動作**および**V動作**と呼ばれている．この2つの動作は，整数の値をとるセマフォアに対して実行される．P動作は次の2つの内の1つのことを行なう．セマフォアの値が正のときには，その値から単に1を減ずるだけである．セマフォアの値が0以下のときには，その値から1を減じたうえで，P動作を行なったプロセスは他のプロセスからの信号を待合せなければならない．その結果このプロセスは一時的に不適格となる．この信号は，同一のセマフォアに対して，別のプロセスがV動作を行なうことによって送られてくる．

V動作も次の2つの内の1つのことを行なう．このセマフォアに対してP動作を行なった結果待合せているプロセスが存在しないときには（すなわち，セマフォアの値は非負），セマフォアの値に1を加えて，実行を続行する．待合せているプロセスが存在するときには，セマフォアの値に1を加え，1つのプロセスに信号を送って待合せを中止させる．信号を送られたプロセスは適格となってスケジューリングを受ける．これらの動作の流れ図は図7.3に示されている．

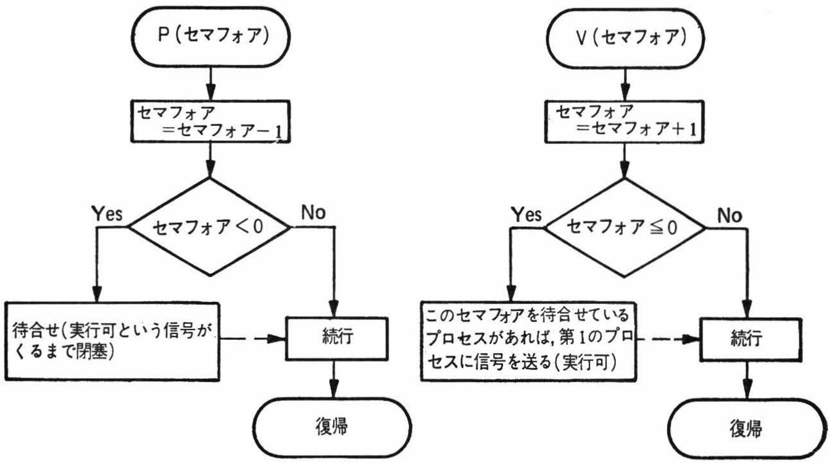


図7.3　PおよびV動作

PおよびV動作をセマフォアと共に用いるのは幾つかの応用が考えられる．たとえば，初期値が1のセマフォアをデータベースに割付けると，そのデータベースのロックとして用いることができる．このデータベースをアクセスするすべてのプロセスが，データベースをアクセスする前にP動作を実行し，アクセスした後にV動作を実行するならば，いかなる場合でもただ1つのプロセスだけにこのデータベースの参照を認めることができる．したがって，このデータベースの有効性を保証することができる．

このモジュールは，PおよびV動作といったスーパバイザ呼出しによって，あるいは時計割込み（割当て時間経過）によって呼出すことが可能である．

7.4.2 記憶管理モジュール

このモジュールは，記憶域の動的割付けおよび解放に必要な動作を行なう．このモジュールは，ジョブの区画割付けにも，オペレーティング・システムが使用する領域の割付けにも使用される．このモジュールは，要求に応じて，所定の語境界（たとえば，2語境界）から必要容量の記憶ブロックを割付けたり，指定された番地の特定の容量の記憶ブロックを解放するルーチンを提供する．その処理のために，このモジュールでは，最適適合アルゴリズムを使用し，“空き”記憶域の一覧表を管理し，要求が処理されるときにこの表から取出したり追加したりする．このモジュールとの通信はスーパバイザ呼出しによって行なわれる．

記憶管理モジュールは，幾つかの目的のために，プロセス管理の下位モジュールを使用する．同じ記憶ブロックが間違って2つのプロセスに割付けられることのないように，記憶管理は，前説で述べたスーパバイザ呼出しPを用いてデータベースを“ロック”する．記憶域が割付けられ表が正しく更新されるとロックが“解かれ”，他のプロセスでこのデータベースを参照することができるようになる．

7.4.3 プロセス管理，上位モジュール

このモジュールは，プロセスを制御するルーチンや（たとえば，プロセス生成や抹消），バッファに入れられたメッセージを送受信するプロセス間通信のルーチンを提供する．このモジュールとの通信は呼出しによって行なわれる．

このモジュールは，プロセス間通信の手段を提供する．プロセスは，k文字のメッセージをある名前のプロセスに送信することを要求してスーパバイザを呼出すことができる．また，プロセスは，送られてきたテキストと送信者の名

前を読むためにこのモジュールを呼出すこともできる．メッセージの長さは任意で，1つのプロセスで読取られるのを待合せるメッセージの個数も任意である．

上の説明でわかるように，この段階でプロセスに名前をつけることが必要となる．名前はプロセスの生成時につけられ，この上位モジュールの中でプロセスを参照するために使用される．前に述べたように，このモジュールではプロセス群という考え方をも導入している．プロセス群は，システム内の1つのジョブに関連するプロセスの集合をいう．

この上位モジュールは，プロセス管理の下位モジュールと記憶管理モジュールの2つに直接関係する．このモジュールは，メッセージ処理において送信者と受信者とを同期させるために，PおよびV基本機能を使用する．また，このモジュールは呼出し者のプロセスの中で実行されるので，トラフィック制御とも間接的に関係がある．プロセスに関係するシステム情報のための記憶域や，メッセージを格納する一時使用のバッファを割付けたり解放したりするために，記憶管理のモジュールが呼出される．

7.4.4　装置管理のモジュールとプロセス

このモジュールは，外部の装置に対して適当な入出力コマンドを出すために必要なルーチンを提供する．上層の他のスーパバイザ・プロセスのモジュールや利用者プログラムと同様に，このモジュールは別個のプロセスによって実行される（本章の場合には装置ごとに1つのプロセス）．このプロセスとの通信は送信されたメッセージによって行なわれ，結果の状態は返送されるメッセージによって通知される．

必要なときには，このメッセージ機能を用いて入出力命令を出すために十分な情報が装置管理プロセスに与えられる．このモジュールの主たる目的は，入出力を起動し，割込みが発生した後に，メッセージ返却のために状態情報を解釈することである．

このモジュールは，2つのプロセスが同時に同一の装置にアクセスしようとすることを防ぐためにセマフォアを使用するので，プロセス管理の下位モジュールと直接関係する．装置に関する情報を納めたブロックに対するロックとして1つのセマフォアが使用される．別のセマフォアが割込みを待合せるため使用され，初期値0のセマフォアにP動作が実行される．

装置管理の特別な部分では割込みを処理する．入出力割込みが発生すると制御は直ちに割込みハンドラに移され，割込みハンドラは割込み符号から装置機

番を求めて対応する装置制御ブロック（UCB）を求める．UCB には，装置管理プロセスが入出力完了を待合せるために使用するセマフォアが納められている．割込みハンドラはこのセマフォアにV動作を実行すると，次には単に，入出力旧 PSW を再びロードして割込み前の状態に復帰する．このようにして，入出力割込みはV動作に変換される．

この例題オペレーティング・システムの装置管理は単純である．その理由は2つある．第1は，すべての入出力装置は専用されており，共用の装置や仮想装置は存在しない．第2は，使用されている入出力装置はカード読取り機と印刷機でマルチプレクサ・チャネルを通じて動作するものだけである．したがって，これらの装置は独自のチャネルを有しているものとして取扱うことができ，入出力スケジューラを用いる必要がない．

7.4.5 スーパバイザのモジュールとプロセス

このモジュールはジョブ・スケジューラとして働く．このモジュールは，各ジョブ流ごとに1つのプロセスとして実行される（したがって，ジョブ群ごとに1つのプロセス）．このモジュールは，利用者ジョブを処理するためのインターフェースを作成するのに，前述のモジュールの機能をすべて利用することができる．

スーパバイザ・プロセスは，各ジョブに順番に，$JOB カードで指定された容量の区画を割付ける．このプロセスは適当な装置管理プロセスを生成して起動させる．次に，利用者の目的デックを区画にロードし，プロセスを生成して，その区画の中でプロセスを起動させる．これらのことがすべてすむと，利用者ジョブが終了するまでは，スーパバイザ・プロセスは必要でなくなる．このプロセスは実行を停止して，利用者プログラムから完了を通知するメッセージが送られてくるのを待合せる．完了が通知されると，スーパバイザ・プロセスは，このジョブのために生成していたプロセスを破壊して取払い，割付けられていた区画を解放してから入力流の中の次のジョブの処理にかかる．

7.4.6 利用者プログラムとプロセス

利用者プログラムは，スーパバイザ・プロセスによって，このプログラムに割付けられた記憶域の区画の中にロードされる．最初は，1個のプロセスが利用者ジョブのために生成されるが，このプロセスは並列処理のために別のプロセスを生成することができる．ジョブが完了したとき，完了の状況が成功か不成功かを利用者プログラムが通知するための基本機能が用意されている．

利用者プログラムは，プロブレム状態で，0でない保護キーを与えられて実行される．このような制限の下で，このプログラムは，システムの他のモジュールが提供する機能の大部分を利用することが可能である．PおよびV動作などの核部分のルーチンは，利用者プログラムでは制限されて利用できない．プロセス間通信機能（プロセス管理上位ルーチン）は，PおよびV動作と同様の同期を行なうために利用することができる．

7.5　核部分のデータベースとルーチン

以下の節では，これまで述べてきたオペレーティング・システムの各段階を詳細に述べる．完全な情報を得たい読者は本章末に示した実際のプログラム・リストを調べてほしい．

本節では，各段階の主要なデータベースをすべて示す．データベースはPL/I 流の構造宣言によって記述され，その後に各部の詳しい説明が続いている．データベースは，オペレーティング・システムの主要な成分である．データベースが注意深く設計されていれば，それらを実際に処理するルーチンを作成することは簡単なこととなる．

7.5.1　スーパバイザ呼出しハンドラ

幾つかのルーチンは，本書の階層構造にうまく適合しない．その最たるものは，拡張計算機の命令の実行を開始させる段階の間で制御を移行させるためのスーパバイザ呼出しハンドラである．IBM 370 のスーパバイザ呼出し (SVC) 命令は，1バイトの演算数を持っている．このバイトは，実行すべき動作の符号を格納するために使用される（たとえば，SVC C'P', SVC C'V'）．SVC 命令が実行されると，SVC 割込みが発生し，制御は SVC ハンドラ・ルーチンに移される．このハンドラは，SVC 符号を調べて，正しい動作に対応しているかどうか，および呼出し者がその動作を要求することが認められているかどうかを検査する．（動作によっては利用者プログラムで実行できないものがある．）レジスタが退避された後に，PSW をロードする命令を用いて，指定されたルーチンに制御が移される．

7.5.2　核ルーチンの一覧表

図 7.4 に核ルーチンの一覧表を示す．各ルーチンについてSVC 符号および機能が示されている．これに加えて，この一覧表には次の項目が示されている：

SVC 符号	機能	ルーチン	退避領域	割込みの可否	利用者呼出しの可否
P	P 同期基本機能	XP	S	NI	NU
V	V 同期基本機能	XV	S	NI	NU
.	トラフィック制御に入る	XPER	I	NI	NU
!	SMC 部分に入る	XEXC	I	NI	NU
,	SMC 部分から出る	XCOM	I	NI	NU
A	記憶域を割付ける	XA	M	I	NU
E	作業域を割付ける	XE	M	I	NU
F	記憶域を解放する	XF	M	I	NU
B	FSB の鎖にブロックを結合する	XB	I	NI	NU
C	プロセスを生成する	XC	F	I	U
D	プロセスを破壊する	XD	F	I	U
H	ジョブを停止し，スーパバイザに通知する	XH	F	I	U
I	PCB を鎖につなぐ	XI	I	NI	NU
J	PCB を鎖から外す	XJ	I	NI	NU
N	名前を与えられてPCB を探す	XN	I	NI	U
R	メッセージを読取る	XR	F	I	U
S	メッセージを送出する	XS	F	I	U
Y	プロセスを開始する	XY	F	NI	U
Z	プロセスを停止する	XZ	F	I	U
?	このジョブを異状終了する	XQUE	I	NI	U

図 7.4 SVC 動作の一覧表

1. 処理を行なうルーチン
2. 使用する退避領域

   I＝割込み退避領域

   F＝割出し退避領域

   M＝記憶退避領域

   S＝システム・セマフォアの退避領域
3. このルーチンは割込み可か否か

   I＝割込み可

NI＝割込み不可

4.　このルーチンは利用者から呼出し可能か否か

U＝利用者が呼出し可能

NU＝利用者から呼出し不可能

退避領域は，1つのモジュールが他のモジュールを呼出すときに，重複しないように割付けられている．以下のルーチンの説明で，トラフィック制御に入るような呼出し，すなわち XP および XPER は，プロセス間で制御を交換する場合には I 退避領域を使用しなければならないことに注目してほしい．システム・セマフォア退避領域以外の退避領域はすべて，プロセスごとに存在する（プロセス制御ブロックの説明を参照）．

これらのルーチンはすべて，無制限に記憶装置参照を可能とするように，0 のキーで実行される．ルーチン XP, XV, XB, XI, XJ, XN, XY, XPER, XEXC, XCOM, および XQUE は割込み禁止（マスクがオフ）で実行され，それ以外は割込み可で実行される．割込みを禁止することによって，そのルーチンが動作中にはプロセスが実行不可能となることを保証している．

割込み可で実行されるルーチン（XA, XE, XF, XH, XC, XD, XR, XS, および XZ）は，このルーチンを呼出すときに XEXC ルーチンを呼出し（“システムは完了しなければならない (SMC)”というモードになる），復帰するときに XCOM ルーチンを呼出す（“システムは完了しなければならない”というモードから出る）．これは，あるプロセスが重要なシステム・ロックを設定している間は，破壊されないということを保証するためである．

直接利用者から呼出せるのはごく一部のルーチンだけである．それらは，XC, XD, XH, XN, XR, XS, XY, XZ, および XQUE ルーチンである．残りのものは，他のシステム・ルーチンによってのみ使用される．これらのプロセス管理の上位モジュールは利用者が呼出しても安全である．その他のルーチンを直接呼出せるようにすると，安全性を破壊する．

引数は，レジスタ 2 で指される領域に入れて核ルーチンに引渡される．返却すべき値もこの引数表に入れなければならない（たとえば，SVC C‘A’で割付けられた記憶番地）．

7.6　処理装置管理のデータベース（下位モジュール）

処理装置管理（下位モジュール）のルーチンとデータベースはたいていのオペレーティング・システムのルーチンによって使用される．プロセス制御ブロ

ック（PCB）とセマフォアは特に重要なデータベースである．

7.6.1 プロセス制御ブロック

プロセス制御ブロック（PCB）はプロセスに関連する情報を格納する．システム中の各プロセスには1つずつ PCB がある．PCB は次のように定義される：

```
DECLARE
1 PCB BASED (PCBPTR) ALIGNED,
  2 NAME CHARACTER (8),
  2 NEXT_PCB_THIS_GROUP POINTER,
  2 LAST_PCB_THIS_GROUP POINTER,
  2 NEXT_PCB_ALL POINTER,
  2 LAST_PCB_ALL POINTER,
  2 STOPPED BIT (1),
  2 BLOCKED BIT (1),
  2 IN_SMC BIT (8),
  2 STOP_WAITING BIT (1),
  2 MESSAGE_SEMAPHORE_COMMON LIKE (SEMAPHORE),
  2 MESSAGE_SEMAPHORE_RECEIVER LIKE (SEMAPHORE),
  2 FIRST_MESSAGE POINTER,
  2 NEXT_SEMAPHORE_WAITER POINTER,
  2 STOPPER_SEMAPHORE LIKE (SEMAPHORE),
  2 STOPPEE_SEMAPHORE LIKE (SEMAPHORE),
  2 AUTO_STORAGE_SIZE FIXED BINARY (31,0),
  2 AUTO_STORAGE_ADDRESS POINTER,
  2 INTERRUPT_SAVE_AREA LIKE (SAVE_AREA),
  2 FAULT_SAVE_AREA LIKE (SAVE_AREA),
  2 MEMORY_SAVE_AREA LIKE (SAVE_AREA);
```

各部分の説明を以下に示す：

NAME：　この部分はプロセスの名前を示す8字の文字列を納めている．名前はこのプロセスを生成するプロセスによって与えられる．この名前は，プロセス管理上位モジュールのルーチンを呼出すときにプロセスを参照するために使用される．

NEXT_PCB_THIS_GROUP：　これは同一のプロセス群における次の PCB へのポインタである．システム内のプロセスは，プロセス群として互に共通部分を持たない集合に分割される．各群に属するプロセスは円環状のリストに連鎖される．このポインタはそのリストの前向きポインタである．1つの群の中の名前はすべて一意的な名前であり，プロセスの名前づけは，つねにその群に相対的に行なわれる．

LAST_PCB_THIS_GROU：　これは，この群の中の PCB の鎖の後向けポインタである．

NEXT_PCB_ALL：　上述のように群によって連鎖されているうえに，システム中のすべての PCB は，前とは独立に，1つの円環リストに連鎖されている．これは，群とは無関係のプロセス管理の下位モジュール（特に，トラフィック制御）に対して設けられている．プロセス群はプロセス管理の上位モジュールの概念である．このポインタは鎖の中の前向けポインタである．

LAST_PCB_ALL：　これは，すべての PCB の鎖の後向けポインタである．

STOPPED：　対応したプロセスが停止していないとき，このビットは0で，停止しているとき，1である．プロセスが停止しているときには，このプロセスは，トラフィック制御から実行可能とはみなされない．プロセスが最初生成されたときには，このプロセスは停止状態である．このプロセスは，“プロセス起動”の基本機能を他のプロセスによって実行することにより起動することができる．非停止状態のプロセスは，同一群の他のプロセスが“プロセス停止”基本機能を実行することによって停止させることができる．これは，通常，プロセスを破壊する準備として行なわれる．

BLOCKED：　対応したプセスが閉塞していないとき，このビットは0で，閉塞しているとき，1である．プロセスが閉塞しているときには，このプロセスは，トラフィック制御から実行可能とはみなされない．通常，プロセスは閉塞されていないが，値が0以下のセマフォアに対してP動作を実行すると閉塞状態となる．このプロセスは，他のプロセスがそのセマフォアに対してV動作を実行すると閉塞状態から解除される．

IN_SMC：　プロセスが“SMC”部分（“システムは完了しなければならない”という部分）を実行しているとき，このバイトは0でなく，SMC 部分を実行していないとき0である．プロセスが SMC 部分に入っているときには，プロセスを停止させようとする動作は，この部分から外へ出るまで延期される．SMC 部分に入ったり，ここから出たりするための基本機能が存在する．

SMC 部分は入れ子の関係になっていることがあるので，このバイトの値は入れ子の深さを示している．

STOP_WAITING： このビットは，停止要求が出ていて，このプロセスが SMC 部分から外へ出るのを待合せているとき1であり，停止要求が待合せていないとき0である．このビットは“プロセス停止”基本機能によって設定される．

MESSAGE_SEMAPHORE_COMMON： これは初期値が1のセマフォアである．他のプロセスがこのプロセスにメッセージを送ろうとするときや，送信されたメッセージをこのプロセスが読出そうとするときには，FIRST_MESSAGE によって指されているメーセージの鎖を探したり変更したりしている間，このメッセージの鎖をロックするためにこのセマフォアが使用される．

MESSAGE_SEMAPHORE_RECEIVER： このセマフォアはメッセージ受信を調整するために使用される．これの初期値は0である．メッセージがこのプロセスに送信されてくるときには，送信プロセスはこのセマフォアに対してV動作を実行して，その値に1を加える．このプロセスに送られたメッセージを読出そうとするときには，このセマフォアにP動作が実行される．したがって，読出すべきメッセージが存在しないときには，“メッセージ読出し”ルーチンはメッセージが送られてくるまで待合せを行なう．

FIRST_MESSAGE： これは，このプロセスによって読出されるのを待合せているメッセージの先頭を指すポインタである．読出すべきメッセージが存在しないとき（すなわち，MESSAGE_SEMAPHORE_RECEIVER の値が0）には，このポインタの値は何の意味もない．

NEXT_SEMAPHORE_WAITER： 同一のセマフォアに関して待合せを行なっているプロセスはすべてリストに連鎖されている．このリストの先頭はセマフォアから指されている．そして，このリストに入っている PCB は，この部分を用いて連鎖されている．このリストの長さは0か，またはセマフォアの値が負のときはその絶対値である．

STOPPER_SEMAPHORE： このセマフォアは，SMC 部分に入っているプロセスを停止させるときの待合せに使用される．停止を行なおうとするプロセスは，停止させようとしているプロセスの IN_SMC バイトが0でないことを知ると，初期値が0のこのセマフォアに対してP動作を実行する．停止されようとしているプロセスが“SMC 部分から外へ出る”基本動作を実行しようとして STOP_WAITING ビットが設定されていることを発見すると，このセマフォアに対してV動作を実行し，次に，STOPPEE_SEMAPHORE に対し

てP動作を実行する．この状況はさらに詳しく第7.11.10節にて説明する．

STOPPEE_SEMAPHORE：　初期値として0を持つこのセマフォアは，SMC 部分から外へ出るまで停止することを延期していたプロセスを停止させるのに用いられる．プロセスはこのセマフォアにP動作を実行して，停止処理のプロセスが停止要求を完了させるまで自分を閉塞させる．

AUTO_STORAGE_SIZE：　この部分は，SVC C'E' 動作によってプロセスに割付けられた特別の記憶域のバイト数を格納している．この情報は，プロセスが破壊されるときに，記憶域が正しく解放されるようにするため使用される．記憶域が割付けられていない場合には，この値は0である．

AUTO_STORAGE_ADDRESS：　このポインタは，SVC C'E' によって割付けられた特別領域の先頭番地である．この情報は，プロセスが破壊されるときに，記憶域を解放するために使用される．このような記憶域が割付けられていないときは，値は0の初期値のまま変化せずに残っている．

INTERRUPT_SAVE_AREA, FAULT_SAVE_AREA, MEMORY_SAVE_AREA：　これらは退避領域である．これらの書式はすべて同一で次の退避領域の節にて述べる．これらは，スーパバイザ・ルーチンが呼出されたとき，レジスタ，PSW, およびその他の状態情報の写しを退避するために使用される．各場合にどの退避が使用されるかについては図7.4に示してある．

7.6.2　退避領域

システム・ルーチンが，ルーチンから出るとき値を復元するために，ルーチンに入ったときの状態情報を格納する退避領域を必要とすれば PCB の中の3つの退避領域の内の1つ，またはシステムの退避領域（PまたはV動作のときに限る）が使用される．退避領域は次のような形となっている：

```
DECLARE
1 SAVE_AREA BASED (SAVE_AREA_POINTER),
  2 OLD_PSW LIKE (PSW),
  2 OLD_REGISTERS (0:15) LIKE (REGISTER),
  2 TEMPORARY (3) BIT (32);
```

各部分は次のように使用される：

OLD_PSW：　これはルーチンに入ったときの旧 PSW である．

OLD_REGISTERS：　これらは，ルーチンに入ったときの汎用レジスタの値である．

TEMPORARY： これは，3語の一時記憶域である．これは，主として，他のルーチンを呼出すときに必要な引数表（主記憶に格納しておく必要のあるもの）を構成するために使用される．ルーチンが3語以上の一時記憶域を必要とする場合には，この3語を用いて記憶域割付けルーチンを呼出すことができる．

7.6.3 セマフォア

セマフォアは，プロセスの基本的な下層の段階の同期のために使用される．セマフォアはPまたはV動作によってアクセスすることができ，その値を直接検査することもできる．その書式は次のとおりである：

```
DECLARE
1 SEMAPHORE BASED (SEMAPHORE_POINTER),
  2 VALUE FIXED BINARY (31,0),
  2 FIRST_WAITER POINTER;
```

各部分は次のように使用される：

VALUE： これはセマフォアの値である．P動作はつねにこの値から1を減じ，V動作はつねに1を加える．セマフォアの初期値は負であってはならない．

FIRST_WAITER： 負の値をとるセマフォアに対しては，このポインタは，NEXT_SEMAPHORE_WAITER ポインタで連鎖されている PCB リストの先頭ブロックを指している．リストの長さはセマフォアの値の絶対値である．0以下の値を持つセマフォアに対してP動作を実行すると，そのプロセスの PCB がリストの最後につながれる．また，負の値を持つセマフォアに対してV動作を実行すると，最初の PCB がリストから取除かれ，このポインタが変更される（この後でこのプロセスは目覚めさせられる）．したがって，FIFO の行列を作ることができる．

7.6.4 RUNNING

これは，現在実行中のプロセスの PCB を指すポインタである．このポインタはトラフィック制御によって設定され，“現在”のプロセスに対する PCB にアクセスする必要があるときに使用される．

7.6.5 NEXTTRY

これは，トラフィック制御が次に実行しようとするプロセスの PCB を指すポインタである．このポインタは，RUNNING によって指される PCB の中の NEXT_PCB_ALL ポインタと同じ値に，トラフィック制御によって設定される．しかし，このポインタはV動作の実行によって変更される．

7.6.6 NEXTTRY_MODIFIED

このビットは，トラフィック制御によって設定された NEXTTRY が変更されていないときには0であり，変更されたときには1である．

7.6.7 SYSTEM_SEM_SAVE_AREA

これは，プロセスを切替えるときに処理装置管理（下位モジュール）によって使用される退避領域である．これが使用される場合については図7.4に示される．

7.6.8 データベースの例

処理装置管理の主要なデータベースが実際の動作でどのようになっているかの例を示したのが 図 7.5 である．図を簡単にするために，PCB の一部の部分のみを示している．

例には，合計7つのプロセスから成る2つのプロセス群と，3つのセマフォアが示されている．JOB1，IN1，および OUT1 が1つのプロセス群を構成し，JOB2，JOB2SUB，IN2，および OUT2 がもう1つのプロセス群を構成している．IN2は現在実行中のプロセスであり，OUT2が次に実行しようとするプロセスの候補である．プロセス IN1，JOB2，および JOB2SUB はセマフォアによって待合せを行なっており，“閉塞”されている．IN1 は，セマフォアAに対するV動作を待合せている．JOB2 と JOB2SUB は共に，セマフォアBを用いて待合せている．V動作が1回セマフォアBに対して実行されると，プロセス JOB2 SUB は“実行可”にされるが，プロセス JOB2 は“B”を待合せている間依然として“閉塞”されたままである．

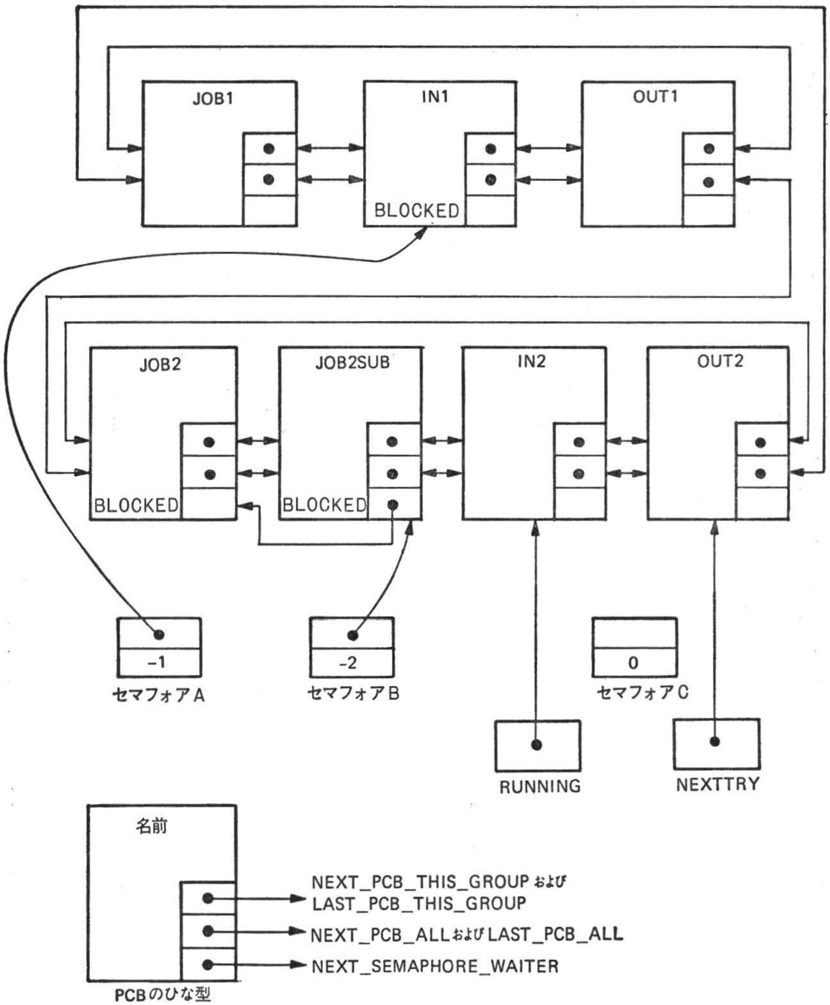


図 7.5 処理装置管理のデータ・ベースの例

7.7 処理装置管理のルーチン（下位モジュール）

7.7.1 トラフィック制御

トラフィック制御は，プロセスのための環境を作るのに用いられる．トラフ

ィック制御は，あたかも，“プロセスの間”で実行される．トラフィック制御へは，時計割込みが検出されたとき，0以下の値を持つセマフォアに対してP動作が実行されたとき，あるいは特別な場合として，SVC C‘.’ (XPER) によって制御が放棄されたとき，制御が渡される．

トラフィック制御は，処理装置多重化のための基本動作を遂行し，その時点で実行可能のプロセス間で計算機の処理装置を分配する．PCB の STOPPED および BLOCKED の両方のビットが共にオフであれば，このプロセスは実行可であるといわれる．プロセスに制御が移されると，割当て時間を使いつくすまで（時計割込みはプロセスの実行を始めて30ミリ秒後に発生するように設定される）あるいは他のプロセスと同期をとるために待合せる必要が生じるまで（0以下の値を持つセマフォアに対してP動作が実行される），あるいは特別な呼出しを行なってトラフィック制御を放棄するまで，実行される．トラフィック制御の全体的な流れ図は 図 7.6 に示されている．

トラフィック制御が呼ばれると制御は直ちにルーチン XPER に移される．このルーチンは，キーの値が0で，割込みを禁止して実行される．レジスタの値や旧 PSW は，PCB の中の INTERRUPT_SAVE_AREA に格納される．

トラフィック制御は，NEXTTRY によって指されているプロセスから開始して NEXT_PCB_ALL の鎖をたどり，システム中のすべての PCB の内で停止したり閉塞したりしていない最初の PCB を求める．このような PCB が存在すると，トラフィック制御は，ポインタ RUNNING がこの PCB を指すように設定し， NEXTTRY を “ALL” の鎖において次に位置している PCB を指すように設定し，30ミリ秒経過すると時計割込みが発生するようにタイマを設定し，標準のサービス・ルーチンを呼出して RUNNING の指す PCB の中の INTERRUPT_SAVE_AREA からレジスタと PSW を復元し，この新プロセスの実行を開始する．

“ALL” の鎖の中の PCB がすべて停止していたり閉塞していたりする場合には，トラフィック制御は，番地部を XPER に，待合せビットをオンに，そして割込みを可能にした PSW をロードする．後刻，入出力割込みが発生すると，旧 PSW の待合せビットをオフとしてから再びロードする．このようにして，トラフィック制御は，入出力割込みが発生するまで待合せてから次のスケジューリングを試みる．このような場合には，割込みが発生するまで実行可のプロセスが存在せず，割込みが発生すると実行可のプロセスが1つ存在することになる．

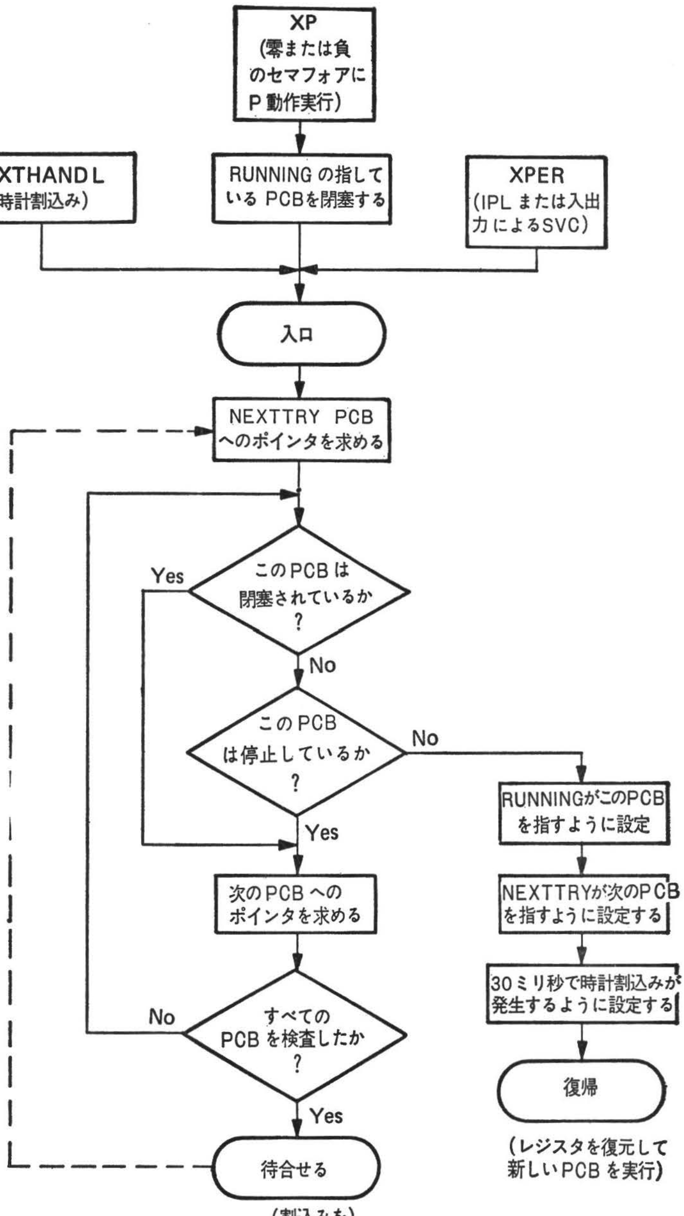


図 7.6 トラフィック制御

7.7.2 ルーチン XP（P 動作）

このルーチンはP基本機能を組込んでいる．呼出されたとき，レジスタ2がセマフォアを指しているものと仮定している．このルーチンは，セマフォアの値から1を減じ，その結果が負でなければ復帰する．

しかし，結果の値が負であれば，このプロセスの PCB はこのセマフォアについて待合せを行なっているプロセスのリストの末尾に挿入される．SYSTEM_SEM_SAVE_AREA の情報が，PCB の中の INTERRUPT_SAVE_AREA へ転送された後で，制御は直接トラフィック制御に移される．

7.7.3 ルーチン XV（V 動作）

このルーチンはV基本機能を組込んでいる．呼出されたとき，レジスタ2がセマフォアを指しているものと仮定している．このルーチンは，セマフォアの値に1を加える．結果の値が1以上であれば復帰する．

値が0以下である場合には，このセマフォアを待合せているプロセスが存在している．リストから最初のプロセスを取除き，BLOCKED ビットをオフとし，このプロセスが再びスケジュールされるようにする．

この新しく目覚めさせられたプロセスが短時間内にスケジュールされるように，通常，XV ルーチンは，NEXTTRY ポインタがこの PCB を指すようにし，NEXTTRY_MODIFIED ビットをオンにする．しかし，このビットがすでにオンとなっているときには，この処理は行なわれない．このようにすると，あるプロセスが実行中に他のプロセスを多数目覚ました場合，最初に目覚めさせられたプロセスがこのプロセスの次に実行される．ここで XV ルーチンから復帰する．

7.7.4 ルーチン XPER（トラフィック制御の入口）

これは，トラフィック制御への直接的な入口である．この入口は，システムを初起させる初期プログラムのロード・ルーチン，あるいは特殊プロセスを実時間応答でスケジュールする必要がある場合に入出力割込みハンドラに限って使用される．

7.7.5 ルーチン XEXC（SMC 部分の入口）

このルーチンは，SMC 部分に入ろうとしていることを通知するために使用

される．このルーチンは，PCB の中の IN_SMC バイトの値を1だけ増加させて復帰する．

7.7.6 ルーチン XCOM（SMC 部分の出口）

このルーチンは，SMC 部分から出ようとしていることを通知するために使用される．このルーチンは，IN_SMC バイトの値から1減ずる．この値がまだ正のときはそのまま復帰する．値が0のときには，PCB の中の STOP_WAITING ビットが検査される．オフのときは XCOM から復帰する．

STOP_WAITING ビットがオンのときは，このビットはオフに戻され，STOPPER_SEMAPHORE に対してV動作が実行され，さらに STOPPEE_SEMAPHORE に対してP動作が実行される．

7.8 記憶管理のデータベース

7.8.1 空き領域ブロック

空き領域ブロック（FSB）は，容量の小さいものから順に連鎖されている．主記憶において隣接した FSB は2つの空き領域のままでなく，1つの大きなブロックに合併される．

割付けが要求されると，空き領域リストから1つのブロックが取外される．1つのブロックが解放されると，隣接した空きブロックを合併したうえで，リストに連結される．

FSB の書式は次のとおりである：

```
DECLARE
1 FREE_STORAGE_BLOCK BASED (FSB_POINTER),
  2 NEXT POINTER,
  2 SIZE FIXED BINARY (31, 0),
  2 UNUSED (FREE_STORAGE_BLOCK. SIZE_8) CHAR (1);
```

各部分は次のように使用される：

NEXT： これは，FSB の容量の小さいものから順に並べられたリストにおいて次の FSB を指すポインタである．鎖の最後に位置する FSB ではこの部分は0（null）となっている．

SIZE： この部分は FSB の容量を示している．この容量はバイト数として

納められている．しかし，記憶管理モジュールに対する要求はすべて2語境界でなければならないので，バイト数は2語の整数倍となっている．また，システム・ルーチンだけが記憶管理を呼出せることに注意してほしい．

UNUSED：　この部分はFSBの残りの部分であり，その大きさは容量部より求められる．

7.8.2　空き領域ブロック・ポインタ（FSBPTR）

このポインタは容量の小さいものの順に並べられたFSBの最初のブロックを指している．鎖にブロックがつながれていない場合には，このポインタは0（null）である．

7.8.3　空き領域ブロック・セマフォア（FSBSEM）

このセマフォアは初期値1を持ち，ロックの役目を果たして空き記憶域リストのアクセスを制御する．記憶管理ルーチンは，空き記憶域リストを検査したり修飾したりする部分に入るときにはこのセマフォアにP動作を実行してこのデータベースの正しい利用を確認し，このデータベースの使用を終えた後にV動作を実行する．

7.8.4　記憶域セマフォア（MEMORY）

このセマフォアは初期値0を持ち，記憶域に対する待合せを制御する．プロセスが記憶域を割付けようとしてそれが不可能であったときには，このセマフォアにP動作を実行して自分自身を閉塞する．他のプロセスが記憶域を解放することによって目覚されたとき，プロセスは再び記憶域を割付けようとする．失敗すれば再び閉塞されることもある．この動作は要求が満足されるまで繰返される．

記憶ブロックが解放されるとこのセマフォアに対して数回のV動作が実行される．V動作が実行される回数はこの時点でこのセマフォアを待合せているプロセスの個数に等しい．

7.8.5　データベースの例

記憶管理の主要なデータベースの例は 図 7.7 に示されている．この例では主記憶容量は20Kバイトである．オペレーティング・システム，システム管理表，および利用者プログラムは13Kバイトを占めている．残り7Kバイトの空き領域は1K，2K，および4Kバイトの3つの部分に分割されている．これら

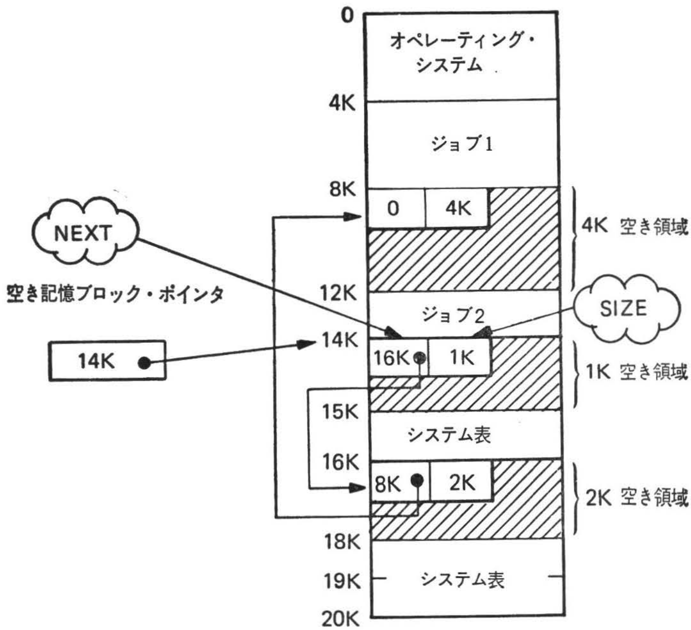


図 7.7 記憶管理のデータベースの例

の部分は FSBPTR と各 FSB 次ブロック・ポインタを用いて連鎖されている.

記憶管理に 3K バイトの領域を要求すると 4K の空き領域が 3K と 1K の 2 つに分割され, 3K が要求者に渡され, 1K がそのまま空き領域として残る. 空きリストの鎖は, 新しい 1K の領域がすでに存在している 2K の領域より小さいことを反影して再編成することも必要である (この鎖は空き領域の容量の小さいものから順に並んでいることに留意してほしい).

7.9 記憶管理ルーチン

7.9.1 ルーチン XA (記憶ブロックの割付け)

このルーチンは記憶ブロックの割付けを行なう. 呼出されたとき, レジスタ 2 が引数表を指しているものと仮定している. 引数表には, バイト数で示した

ブロック容量，割付けた領域の番地を返却する1語，および要求する割付け境界の種類を示す2の幅乗数を納めている．この数が8ならば，2語境界への割付けであることを示している．

このルーチンは，まずFSBSEMセマフォアにP動作を実行する．次に，空き領域リストを順にたどり，要求の領域を収容できる容量のブロックを探す．余剰の領域を返却して空きリストに再び結合するのに割付けた領域のどちら側にするのがよいかを決定する．分割の個数を小さくするために指定の整合条件を持つ領域は可能な限りブロックの先頭に近く割付ける．このルーチンは，B基本機能を使用して，再結合を実行する．割付けられた領域の番地は引数表の中に挿入され，FSBSEMセマフォアに対してV動作を実行した後で制御を復帰する．

しかし，要求を満足させることができない場合には（すなわち，このルーチンで適当なブロックを見つけることができない内に空き記憶域リストが終了したとき），このルーチンはFSBSEMセマフォアに対してV動作を実行して空き記憶域のデータベースのロックを解き，次にMEMORYセマフォアに対してP動作を実行する．このようにして，別のプロセスが記憶域を解放するまで待合せを行なう．このルーチンは，待合せを行なった後に，ルーチンの最初に復帰して再び割付けを試みる．

このルーチンにはもう1つ別の入口がある．SVC C'E' を実行すると，記憶管理はこのプロセスに対して作業領域を割付ける．この呼出しでは，容量と開始番地をPCBの中に格納するので，各プロセスは1回だけこのような呼出しが認められる．このようにして割付けられた記憶域を持つプロセスが破壊された場合には，この記憶域は自動的に空き記憶リストに返却される．それ以外の点では，この入口から入っても，上述のXAルーチンと全く同様の動作を行なう．

7.9.2　ルーチン XB（FSB の鎖にブロックを連鎖する）

このルーチンは，新しい領域とすでに存在しているFSBとの合併が必要でないものとして，FSBを空き記憶域リストに結合する．XBはXAから直接呼出すことができる．このルーチンが呼出されたとき，レジスタ2は，FSBの鎖につなぐべき領域のバイト容量および領域の番地を納めた引数表を指している．このルーチンは，すでにFSBSEMセマフォアに対してP動作を実行しているXAおよびXFルーチンからのみ呼出されるので，他のプロセスからデータベースへの干渉を心配する必要はない．

このルーチンが空き記憶域リストを探して新しいブロックを挿入すべき位置を求めると，このブロックをリストに結合して FSB の書式を整える．そして制御を復帰する．

7.9.3 ルーチン XF（記憶ブロックの解放）

このルーチンは記憶域を解放し，必要ならば新しい領域と既存の FSB とを併合する．このルーチンが呼出されたとき，レジスタ2は，解放すべき領域のバイト容量と領域の番地とを格納した引数表を指している．

このルーチンは，まず FSBSEM セマフォアにP動作を実行して空き記憶域リストにロックをかける．次に，空き記憶域リストを探して，解放しようとしている領域と隣接している FSB がないかどうかを調べる．そのような FSB が存在する場合には空き記憶域リストから取外し，解放するブロックの容量と番地を計算し直す．リストの終りに達すると，このルーチンはB動作を実行し，ルーチン XB を呼出して空き記憶域リストにブロックを結合する．

MEMORY セマフォアの値を調べて主記憶が解放されるのを待合せているプロセスの個数を求めると，このルーチンは，MEMORY セマフォアに対してその回数だけV動作を実行する．ここで制御は復帰する．

7.10 処理装置管理のデータベース（上位モジュール）

処理装置管理の上位モジュールは，下位モジュールと幾つかのデータベースを共用している．特に，PCB には下位モジュールと上位モジュールの両者が使用する欄が用意されている（たとえば，セマフォアの鎖を示す欄は下位モジュールだけが，プロセス名の欄は上位モジュールだけが使用される）．これに加えて，上位モジュールでもメッセージに関係するデータベースの処理が必要となる．

7.10.1 メッセージ

メッセージは，プロセス管理上位モジュールのデータベースとしてプロセス間通信のために使用される．その構造は次に示すとおりである：

```
DECLARE
1 MESSAGE BASED (MESSAGE_POINTER),
  2 SENDER POINTER,
```

```
2 NEXT POINTER,
2 SIZE FIXED BINARY (31,0),
2 TEXT CHARACTER (MESSAGE.SIZE);
```

各部分は次のように使用される：

SENDER：　これは，このメッセージを送信したプロセスの PCB を指すポインタである．

NEXT：　プロセスから読出されるのを待合せているメッセージはすべて NEXT 部によって連鎖されている．最初のメッセージは PCB の中の FIRST_MESSAGE 部によって指定される．

SIZE：　これは，メッセージ本文の文字数である．

TEXT：　この文字部は，送信者によって与えられたメッセージの本文である．

7.10.2　データベースの例

図 7.8 はメッセージと PCB との関係を示している．この例では，プロセスAに3つのメッセージ——2つはBから，1つはCから——が送られている．A が SVC‘メッセージ受取り’を実行すると，B から送られた HELLO が与えられる．この時点で，MESSAGE_SEMAPHORE_RECEIVER は3から2に減ぜられる．次のメッセージ受取りでは GOOD_DAY が，第3のメッセージ受取りでは LET'S GO が与えられる．さらにメッセージを要求すると，まだメッセージは届いておらず，A が MESSAGE_SEMAPHORE_RECEIVER に P 動作を実行すると値は −1 となり，他のメッセージが送信されてくるまでプロセスを閉塞する．例では，プロセス B はすでにこの状態となっている．B は C に ARE YOU READY ？　というメッセージを送り，答えを待っている．

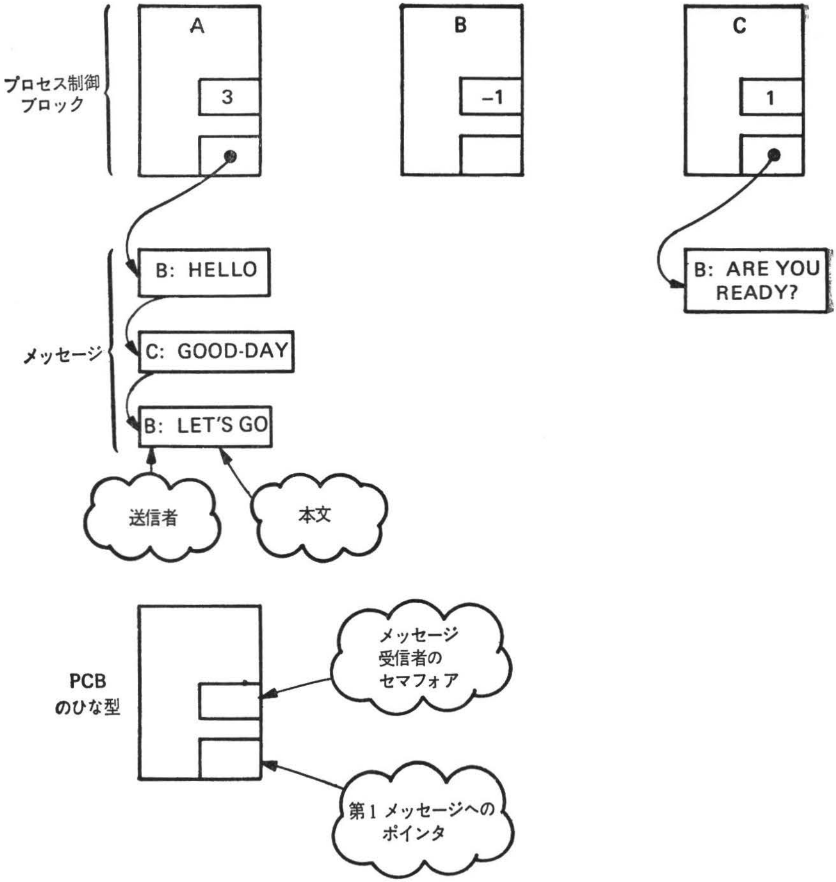


図 7.8 処理装置管理（上位モジュール）のデータベースの例

7.11 処理装置管理のルーチン（上位モジュール）

7.11.1 ルーチン XC（プロセスの創成）

このルーチンは“プロセス創成”の機能を実行する．呼出されたとき，レジスタ2は，創成されるプロセスの名前だけを納めた引数表を指している．

このルーチンは，まずこの名前がすでにグループの中で使用されていないかどうか検査する．もしすでに名前が使用されていれば，ルーチン XQUE を呼出して誤りルーチンに入る．名前がまだ使用されていないならば，ルーチン XA を呼出して新しい PCB を割付ける．新しい PCB にはこの名前が格納され，STOPPED ビットがオン，BLOCKED ビットがオフ，セマフォアが初期値，IN_SMC バイトが0にそれぞれ設定されている．次に，このルーチンは XI ルーチンを呼出して PCB を2つの鎖に連結する．ここで制御を復帰する．

7.11.2 ルーチン XD（プロセスの破壊）

このルーチンは“プロセスの破壊”の機能を実行する．呼出されたとき，レジスタ2は，破壊すべきプロセスの名前のみを格納した引数表を指している．破壊すべきプロセスは，あらかじめ“プロセス停止”機能（ルーチン XZ）によって停止されておかなければならない．

このルーチンは XN ルーチンを利用して，破壊すべきプロセスの PCB の番地を決定する．もし XN ルーチンで，このグループの中に与えられた名前を見出すことができない場合には誤りルーチンが呼出される（ルーチン XQUE)．破壊すべきプロセスは停止状態でなくてはならない．ルーチン XJ は，PCB を2つの鎖から切離し，他のプロセスがメッセージを送信しないようにする．このプロセスによって読出されるのを待合せているメッセージはすべて解放される．次に，特別の作業域の開設を示す PCB の欄を検査し，存在すれば解放する．最後に，PCB の領域を解放し（これらの解放はすべて XF ルーチンで解放される)，XD ルーチンから復帰する．

7.11.3 ルーチン XH（ジョブ停止とシグナル・スーパバイザ）

このルーチンは，“このジョブを停止させよ”という機能を実行する．使用する引数はない．

このルーチンが処理することは，このジョブを停止して平常の状態に戻れというメッセージを（ルーチン XS を通じて）スーパバイザ・プロセス“*IBSUP”に送信することだけである．次に，このルーチンはループを回って，処理を待合せているメッセージを読んでは捨てて最後にすべてのメッセージが解放されたら永久に自分を閉塞させる．

7.11.4 ルーチン XI（PCB を鎖に連結する）

このルーチンは，PCB を2つの鎖に連結する機能を実行する．呼出された

とき，レジスタ2は PCB を指している．

この PCB は，すべての PCB の鎖とこのプロセス・グループの鎖の中に，この実行中のプロセスの PCB の直後に結合される．ここで制御が復帰する．

7.11.5 ルーチン XJ（PCB を鎖から取除く）

このルーチンは，PCB を2つの鎖から取除く機能を実行する．呼出されたとき，レジスタ2は PCB を指すポインタを格納している．

PCB は，リンクを変更して，2つの鎖から取外される．このルーチンは記憶域の解放までは行なわない．解放は呼出者にまかされる．ここで制御が復帰する．

7.11.6 ルーチン XN（名前で PCB を探す）

このルーチンは，プロセスの名前が与えられているとき，このプロセスの PCB を求める機能を果たす．呼出されたとき，レジスタは，名前と PCB へのポインタを格納すべき領域から成る引数表を指している．

このルーチンは，実行中のプロセスの PCB から始めて，NEXT_PCB_THIS_GROUP の鎖をたどり，要求の名前を持つ PCB を求める．それが求まると引数表にポインタを格納して制御を復帰する．要求された名前の PCB が見つからない場合には，ヌル・ポインタを格納して復帰する．

7.11.7 ルーチン XR（メッセージの読込み）

このルーチンは“メッセージ読込み”機能を実行する．呼出されたときレジスタ2は，送信者名の領域，本文を受取るために用意された領域の容量およびこれに続くメッセージ本文を受取るための領域から成る引数表を指している．

このルーチンはまず MESSAGE_SEMAPHORE_RECEIVER に対して P 動作を実行する．もしメッセージがまだ届いていないときにはプロセスは待合せ状態となる．次に，このルーチンは MESSAGE_SEMAPHORE_COMMON に対してP動作を実行してメッセージの鎖にロックをかける．これらのセマフォアはもちろん現在のプロセスの PCB に入っているものである．次に，メッセージ・リストから最初のメッセージが取出されて鎖に対するロックをとく．本文は受信領域に移され，必要ならば，切り捨てや空白の詰め合せが行なわれる．引数表の容量部は転送された有効文字数を示すように更新される．送信プロセスの名前が送信者の PCB から複写され引数表に格納される．最後に，メ

ッセージ・リストの中のこのメッセージの使用していた記憶域が解放され，制御を復帰する．

7.11.8　ルーチン XS（メッセージ送信）

このルーチンは“メッセージ送信”機能を実行する．呼出されたときレジスタ2は，宛先のプロセス名と，本文の文字数およびそれに引続いて本文自体を納めた引数表を指している．

このルーチンはまず与えられた名前でプロセスの PCB を求める．もしそのような PCB が存在しない場合には，ルーチン XQUE を呼出して誤りルーチンに入る．PCB が求まれば，この PCB の中の MESSAGE_SEMAPHORE_COMMON に対して P 動作を実行してメッセージの鎖にロックをかける（これは，メッセージ・ブロックを格納するのに十分な大きさの領域を割付けてから行なわれる）．次に，このメッセージ・ブロックはメッセージ・リストの末尾に連鎖され，メッセージ・リストの長さを PCB の MESSAGE_SEMAPHORE_RECEIVER の値から決定する．このメッセージ・ブロックには，送信者の PCB ポインタ，値がヌルの NEXT ポインタ，本文の文字数，および本文自体が格納される．次に，このルーチンは，MESSAGE_SEMAPHORE_COMMON および MESSAGE_SEMAPHORE_RECEIVER に対して V 動作を実行して復帰する．

7.11.9　ルーチン XY（プロセスの起動）

このルーチンは“停止”状態にあるプロセスを起動させる機能を果たす．呼出されたときレジスタ2は，起動すべきプロセスの名前と開始する記憶番地を納めた引数表を指している．このプロセスのために最初にロードすべきレジスタは SVC 呼出しが起こったとき存在していたものである．これが，新しく創成されたプロセスに引数を渡す方法である．

このルーチンは，まず，与えられた名前のプロセスの PCB に対するポインタを求める．このような PCB が存在しない場合には，ルーチン XQUE を呼出して誤りルーチンに入る．次に，このルーチンは，この PCB の INTERRUPT_SAVE_AREA に指定されたレジスタと指定の番地部を持つ PSW とを格納する．PSW の番地部以外の部分は，このルーチンが呼出されたとき存在していた旧 PSW と同一である．PSW の中の STOPPED ビットはオフとされ，制御は復帰する．

7.11.10 ルーチン XZ（プロセス停止）

このルーチンは“プロセス停止”の動作を実行する．呼出されたときレジスタ2は，停止すべきプロセスの名前だけを納めた引数表を指している．

このルーチンは，まず，与えられた名前のプロセスの PCB に対するポインタを求める．このような PCB が存在しない場合には，ルーチン XQUE を呼出して誤りルーチンに入る．もしこの PCB の IN_SMC バイトが0であれば，このルーチンは単に STOPPED ビットをオンとして復帰する．

もし IN_SMC バイトが0でないときには，プロセスは SMC 部分を実行しているので，STOP_WAITING ビットをオンに設定して STOPPER_SEMAPHORE に対して P 動作を実行する．プロセスが次に実行し始めるときまでに，停止させられるプロセスは SMC 部分から去り，通常のように停止させることが可能となる．

上述のことには1つ例外が存在する．それは，システム・プロセスでないものはシステム・プロセスを停止させることができないことである．（システム・プロセスは*印で始まる名前で区別することができる．）

7.11.11 ルーチン XQUE（異状終了）

このルーチンは，ジョブの異状終了を行なう．このルーチンは XH ルーチンと同様に動作するが，誤りが発生したことを示すメッセージを発生することが異なっている．

7.12 装置管理のデータベース

多くの装置管理モジュールは，利用者プロセスと同時実行されるプロセスとして動作する．これに加えて，幾つかのデータ・ベースは，入出力割込みハンドラおよび装置管理プロセスの両者によって使用される．

7.12.1 装置制御ブロック

装置制御ブロック（UCB）は，システムに接続されているすべての装置（カード読取り機，印刷機など）に対して，恒久的に割付けられた領域の中に格納されている．その書式は次のとおりである：

DECLARE

```
1 UNIT_CONTROL_BLOCK BASED (UCB_POINTER),
  2 ADDRESS FIXED BINARY (31,0),
  2 USER_SEMAPHORE LIKE (SEMAPHORE),
  2 WAIT_SEMAPHORE LIKE (SEMAPHORE),
  2 STATUS BIT (64),
  2 FAST_PROCESSING_REQUIRED BIT (1);
```

各部分は次のように使用される．

ADDRESS：　これは，入出力起動命令の中で使用される物理装置およびチャネルの機番である（たとえば，1台の印刷機は X '000E' である）．

USER_SEMAPHORE：　このセマフォアは初期値1を持ち，1台の装置に2つ以上のプロセスが同時に要求を出さないように，UCB にロックをかける．

WAIT_SEMAPHORE：　このセマフォアは初期値0を持ち，割込みを待合せるために装置ハンドラ・ルーチンによって使用される．このセマフォアに対して P 動作を実行すると（セマフォアの値は負となり），プロセスは閉塞される．割込みが起こると，割込みルーチンは適当な UCB の中の WAIT_SEMAPHORE に対して V 動作を実行する．

STATUS：　この部分は，割込みルーチンによって格納された装置の状態を記録する．ここには，通常この装置に対する最も新しいチャネル状態語が格納される．

FAST_PROCESSING_REQUIRED：　このビットは，迅速な処理が必要とされないとき0で，必要とされるとき1である．迅速な処理が必要とされるときには，割込みルーチンは，WAIT_SEMAPHORE に対してV動作を実行してから，トラフィック制御に制御を移す．トラフィック制御およびV動作の節で述べたとおり，このような処理によって，割込みの待合せを行なったプロセスの実行を直ちに再開することができる．

7.12.2 CAW セマフォア（CAWSEM）

このセマフォアは初期値1を持ち，チャネル番地語（CAW）のロックとして働く．CAW を参照する入出力起動命令を実行するルーチンは，CAW に情報を格納する前にこのセマフォアに対してP動作を実行し，入出力起動命令の実行後にV動作を実行する．

7.13 装置管理ルーチンとプロセス

装置管理は，装置ハンドラ・プロセスと特殊な入出力割込みハンドラに分けられる．装置ハンドラ・ルーチン自体は，装置ごとに特別のプロセスの中に置かれている．これらのプロセスは，元は上層のスーパバイザによって創成され，これらのプロセスとの通信は，プロセス制御の上位ルーチンが提供するメッセージの基本機能によって行なわれる．

入出力割込みの受付けと処理は，入出力割込みが起こったときに呼出される特別のルーチンによって実行される．このルーチンの実行はきわめて短時間であり，状態情報を格納して適当な UCB の中の WAIT_SEMAPHORE に対してV動作を実行する．これだけ行なうとこのルーチンは割込まれたプロセスに制御を復帰する．

7.13.1 読取り機ハンドラ・ルーチンとプロセス

このルーチンはカード読取り機を制御する．このルーチンは，読取り機プロセスのすべてによって共用される．呼出されたとき，レジスタ3に UCB 番地が，レジスタ4に使用すべき保護キーの値が設定されている．次に，このルーチンはメッセージを読込むループに入る．もしメッセージが“READ xxxx”の形式でないときには別のメッセージが読込まれる．ただし，xxxx は4バイトで表わした2進の記憶番地である．

メッセージが正しい形式であれば，このルーチンは，UCB の中の USER_SEMAPHORE に対してP動作を実行し，処理の途中で別のプロセスがこの装置を同時に使用しないようにする．次にこのルーチンは，適当なチャネル・コマンド語（CCW）を納めるチャネル制御ブロックを構成する．CCW で指定される動作は“読込み”要求であり，受信したメッセージの中で指定された番地が用いられている．このメッセージが利用者プロセスから送信されたものであれば，この番地の保護キーが，システムによってこのルーチンに示されたキーと一致するかどうかの検査が行なわれる．CCW の計算機は 80 文字とされ，チェイン・コマンドや特別の割込み指定などのフラグはすべて0に設定される．次に CCW の番地と値が0のキーを持つ CAW が構成される．ここで CAWSEM セマフォアに対してP動作が実行され，CAW が設定され，適当な読取り機に対して入出力起動命令を実行し，CAWSEM セマフォアに対してV動作が実行される．UCB の中の WAIT_SEMAPHORE に対してP動作が

実行され，プロセスは割込みが発生するまで待合せる．

割込みハンドラがこのプロセスを目覚ませた後に，割込みルーチンで格納された状態が検査される．状態が要求の正常完了あるいは異状完了を示している場合には，次に述べるように，プロセスはカードの情報を検査する．もし動作が完了していない場合には，プロセスは再び待合せ状態となる．

プロセスは呼出し者のプロセス名を検査する．名前が＊で始まっているときには，スーパバイザ・プロセスのモジュール（名前づけについてはスーパバイザ・プロセスのモジュールの節を参照）の一部であるとみなされ，要求の完了の状態に応じて OK または NO が返却される．

呼出し者のプロセス名が＊で始まっていない場合には，利用者プロセスとみなされ，状況はもっと複雑となる．

読込んだカードが ＄JOB カードであれば，このプログラムのデータ終了である．したがって，このカードを特別の領域に退避して，この利用者の領域を空白としてから，呼出し者に NO メッセージを返送する．ここでこのプロセスは，システムからの要求を受取るまでは，利用者のすべての要求に対して NO を返すようなループに入る．システムから要求が出されると，指定された領域にカードの情報を格納して OK メッセージを返却する．

7.13.2　印刷機ハンドラ・ルーチンとプロセス

このルーチンは印刷機出力を処理する．このルーチンはすべての印刷機ハンドラのプロセスによって共用される．動作は，読取り機ハンドラの動作と全く類似している．メッセージの書式は PRIN xxxx であり，xxxx は印刷すべき領域の番地を示している．（プロセス名によるデータの検査は行なわれない．）

7.13.3　汎用装置ハンドラ・ルーチン（EXCP）およびプロセス

このルーチンは，装置に対するインターフェースの役割りを果たす．このインターフェースにおいて，利用者は入出力コマンドを準備する．ルーチンの初期化と動作の方法は，上述した読取り機および印刷機ハンドラと類似しているが，このルーチンに送られるメッセージは EXCP xxxxcccc の形式である．ただし，xxxx は装置とチャネルの機番であり，cccc は使用するチャネル・コマンドの先頭番地である．EXCP ルーチンは xxxx を用いて，その装置の UCB を求め，入出力を起動させる．

EXCP ルーチンで駆動した装置に対して割込みが起こると，EXCP インタ

ーフェースは sssssss の書式のメッセージを返却する．ただし，sssssss は装置およびチャネルから返却された状態である．

UCB の中の FAST_PROCESSING_REQUIRED ビットを用いることによって，EXCP の割込みを高速に処理することが可能である．

このルーチンは，再び，利用者のメッセージを待合せる．受取ったメッセージが AGAIN であれば，このルーチンは別の入出力割込みを待合せ，利用者にその状態を返却する．このようにすると，利用者はチャネル完了と装置完了の両方の検査を行なうことができる．メッセージが OK の場合には，このルーチンは UCB を解放して，別の BXCP 型のメッセージを受取ることができるようになる．

7.13.4 入出力割込みハンドラ・ルーチン

入出力割込みルーチンは，入出力要求の後で発生する入出力割込みの処理を実行する．基本的には，このルーチンは，割込みが起こったときに実行していたプロセスの一部として実行される．このルーチンは小数の動作を実行し，割込まれたプロセスの実行を再開する．

まず，割込みを起こした装置の UCB を求める．論理 OR によって UCB の状態部にチャネル状態語 (CSW) を納めてから，UCB の WAIT_SEMAPHORE に V 動作を実行する．通常，この時点でレジスタの値と PSW の値を復元して，このルーチンから復帰する．旧 PSW の待合せビットがオンであったときはつねにオフとされる．(この意味についてはトラフィック制御の節を参照せよ.)

UCB の FAST_PROCESSING_REQUIRED ビットがオンならば，このルーチンは XPER ルーチンを使用して，直接トラフィック制御に制御を移し，新しいプロセスを実行させる．NEXTTRY 部は，通常，装置ハンドラ・プロセスの PCB を指すように変更されているので，このルーチンでは確かに迅速な処理が行なわれる．

7.14 スーパバイザ・ルーチンとプロセス

スーパバイザ・プロセスはジョブのスケジューリングと実行の準備を行なう．1つのジョブ流ごとにスーパバイザ・プロセスが存在する．本書の例では，1台のカード読取り機ごとに1つのジョブ流がある．各スーパバイザ・プロセスは別々のプロセス群に属していて，これらのプロセス間では通信が行な

われない．これらのプロセスは相互に他のプロセスの存在がわからない．各スーパバイザ・プロセスの名前は *IBSUP であり，各 *IBSUP は，ジョブ流の入力および出力装置の処理を行なう2つの副プロセス *IN および *OUT を持っている．

7.14.1 ジョブ流処理ルーチン

$JOB カード（各ジョブの最初のカード）を読込むと，スーパバイザ・プロセスはこのカードをまず印刷機に送る．次に，必要記憶量を決定して，保護のハードウェアの条件に合うように2K語の境界に割付ける．この領域の保護キーは，利用者プログラムに割当てられたキーの値に設定される．次に，USERPROG という名前のプロセスを作る．このプロセスは，利用者プログラムが最初に実行し始めるプロセスとなる．

次に，スーパバイザ・プロセスは，装置割付け指定を調べる．この部分は，名前＝装置種別の形式となっている．（この‘名前’は * で始まってはならない.）各指定に対して，‘名前’というプロセスを作り，この装置種別に対するインターフェース・ルーチンの中で，このプロセスを起動する．IN または OUT については，使用されるルーチンは，利用者プロセスからメッセージを読取ってそのまま *IN あるいは *OUT プロセスに送り，そのプロセスの返答を待合せて呼出しを行なった元のプロセスに返答を返す．EXCP に対しては，第7.13.3節に述べたインターフェース・ルーチンが使用される．

スーパバイザは，次に，利用者の目的デックをその利用者の区画に読込む．必要に応じて再配置が行なわれる．これがすむと，プロセス USERPROG の PCB は，利用者モードで指定されたキーを用いて実行するように変更される．これで利用者プロセスは実行を開始することができる．

USERPROG プロセスを起動させると，スーパバイザ・プロセスはメッセージが返却されてくるのを待合せる．このメッセージは，“成功”あるいは“失敗”とがある．いずれの場合でも，このメッセージの内容は印刷機で印刷され，スーパバイザ・プロセスは，利用者ジョブに対して作ったプロセスや利用者プロセスが作ったプロセスをすべて破壊し，割付けられた記憶域の区画を解放し，次のジョブの処理を始める．

実行すべきジョブがこれ以上存在しない場合には，与えられていないメッセージを読込もうとすることのためにスーパバイザ・プロセスは閉塞状態となる．

ジョブ流処理ルーチンの流れ図は図7.9に示されている．詳細は特に重要で

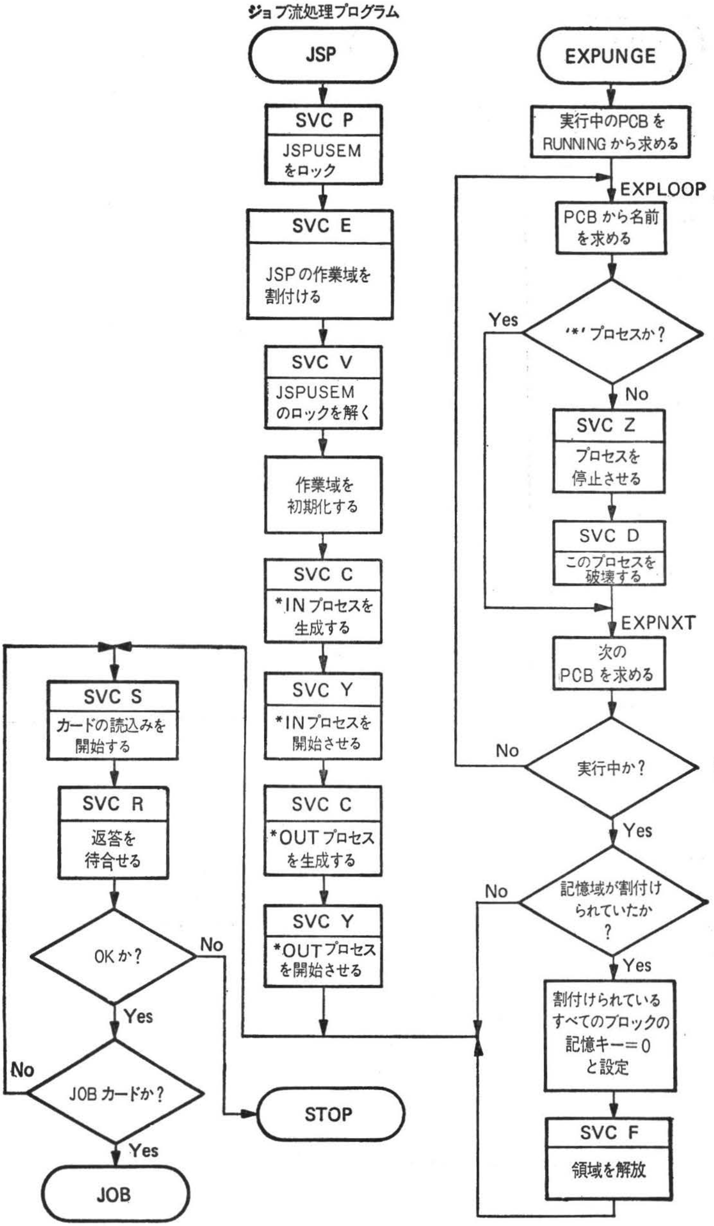


図 7.9 スーパバイザ・プログラムの流れ図（ジョブ流処理プログラム）（その1）

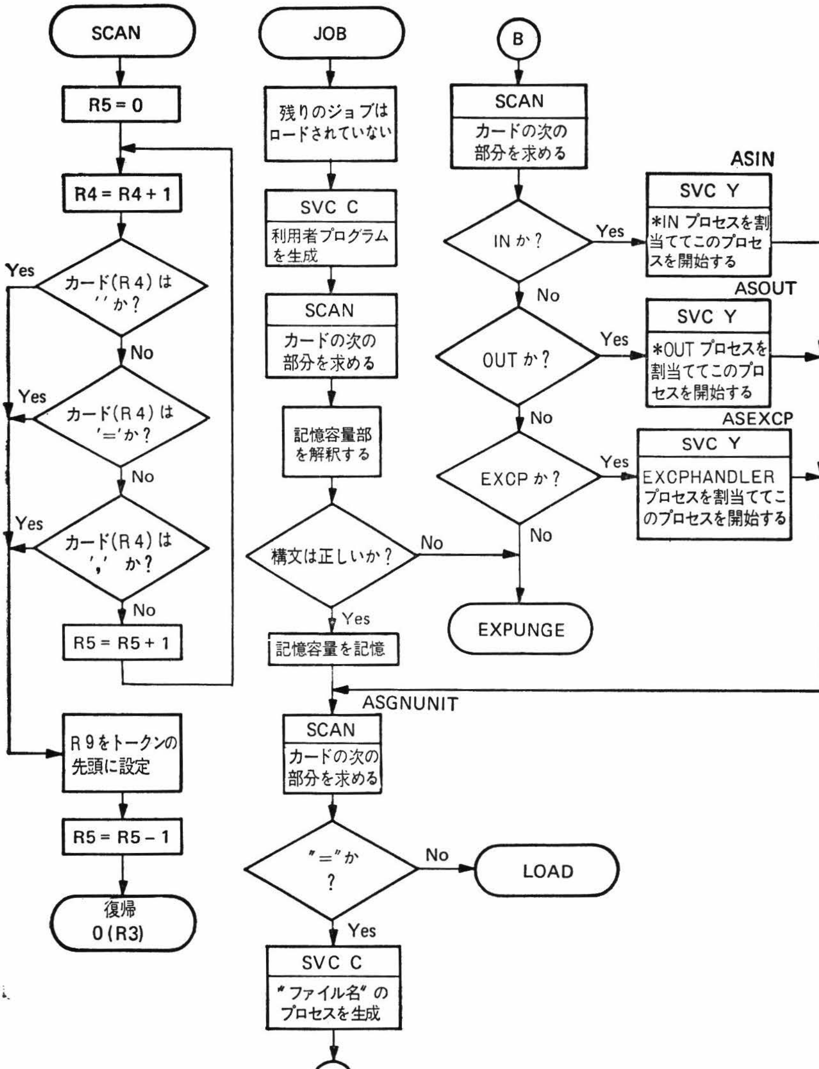


図7.9　スーパバイザ・プロセスの流れ図（ジョブ流処理プログラム）（その2）

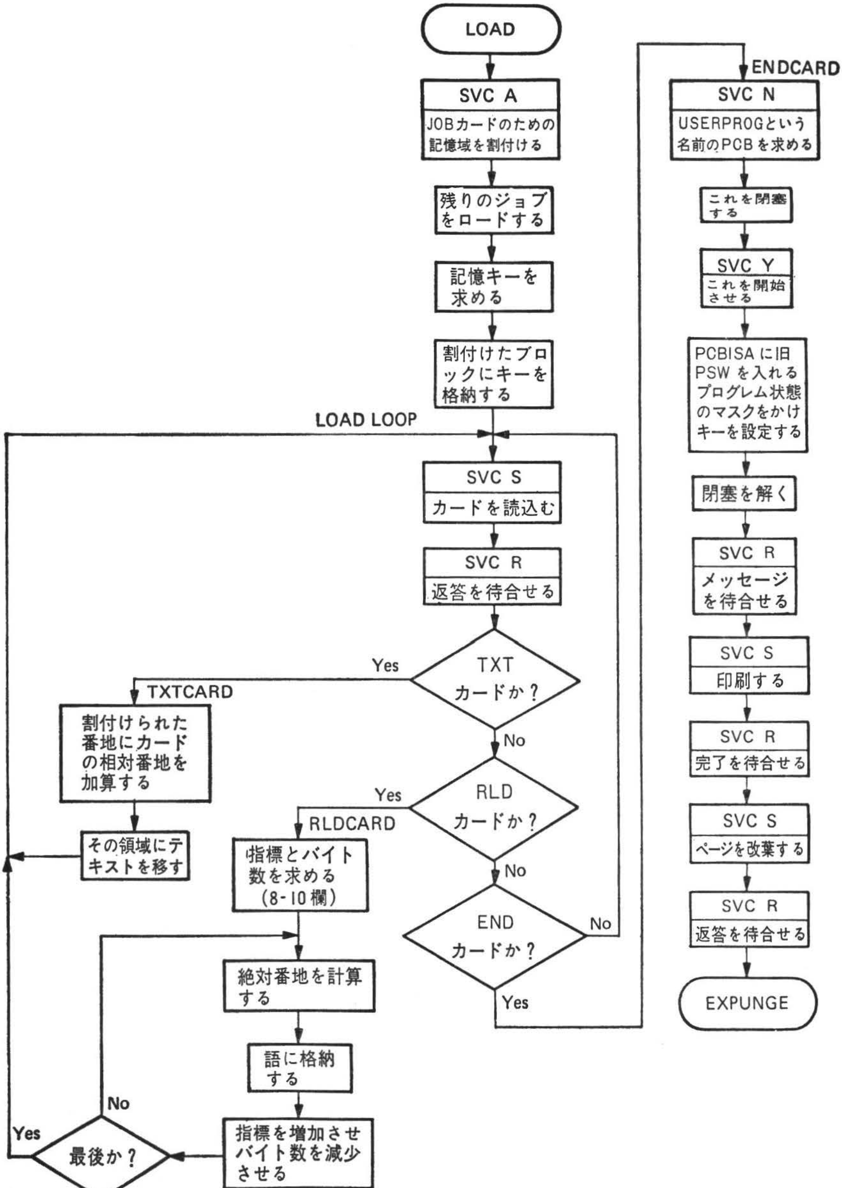


図 7.9　スーパバイザ・プロセスの流れ図（ジョブ流処理プログラム）（その3）

はないが，このルーチンは拡張計算機の多数の機能を用いていることに留意してほしい（たとえば，A, C, D, F, N, P, R, S, V, Y, ZなどのSVC）．利用者の最初のプロセスが起動する前でも，たいていのオペレーティング・システムのルーチンは，ジョブ流処理ルーチンによって使用されていることがわかる．

7.14.2　スーパバイザの初期化

これまで見てきたように，スーパバイザ・プロセスは，システムのすべてのプロセスを創成し制御する．しかし，スーパバイザ・プロセスはどのようにして作られたのであろうか．各種の SVC ルーチンはどのようにして初期化されたのであろうか．オペレーティング・システムの初期化は場合によっては複雑である．システムによっては，この初期化にかなりの時間がかかる．本書のオペレーティング・システムの例では，初期化は初期プログラム・ロード・ルーチン（IPLRTN）によって行なわれる．IPLRTN を含めて，オペレーティング・システムのプログラムは，最初，第2章で述べたような初期プログラム・ロード機構によってロードされる．ロードされると制御は IPLRTN に移される．

IPLRTN は，システムの大部分とは無関係に実行する．IPLRTN には，終了時に論理的にトラフィック制御に入るまでは停止点が存在しない．初期化が完了する前にトラフィック制御に入ってシステムが停止できるように，幾つかのポインタが設定されている．このような制御の移行は，記憶域が小さすぎる場合などに起こる．

初期化ルーチンは，最初，常駐領域に自分自身の PCB を設定する．そして PCB の中の鎖とポインタ RUNNING および NEXTTRY をこの PCB 自身を指すように設定する．次に，空き記憶域（すなわち，システムで使用されない記憶域）は空き記憶域リストに入れられ，すべての保護キーは0に設定される．

この時点で多くの核部分は動作状態となり，使用可能となる．プロセス群に1つずつ入っている各スーパバイザ・プロセスに対して1つの PCB が作られる（したがって，ここでは“プロセス創成”の基本機能を直接利用することはできない）．作られたプロセスは *IBSUP という名前である．IPLRTN は，これらのプロセスに対して“プロセス起動”ルーチンを呼出して，ジョブ流処理ルーチンの中で起動させる．レジスタ3は，読取り機に対する UCB へのポインタと印刷機に対する UCB のポインタを納めた引数表を指し（UCB はすべて，主記憶の常駐部分に格納されている），レジスタ4は，このジョブ流の

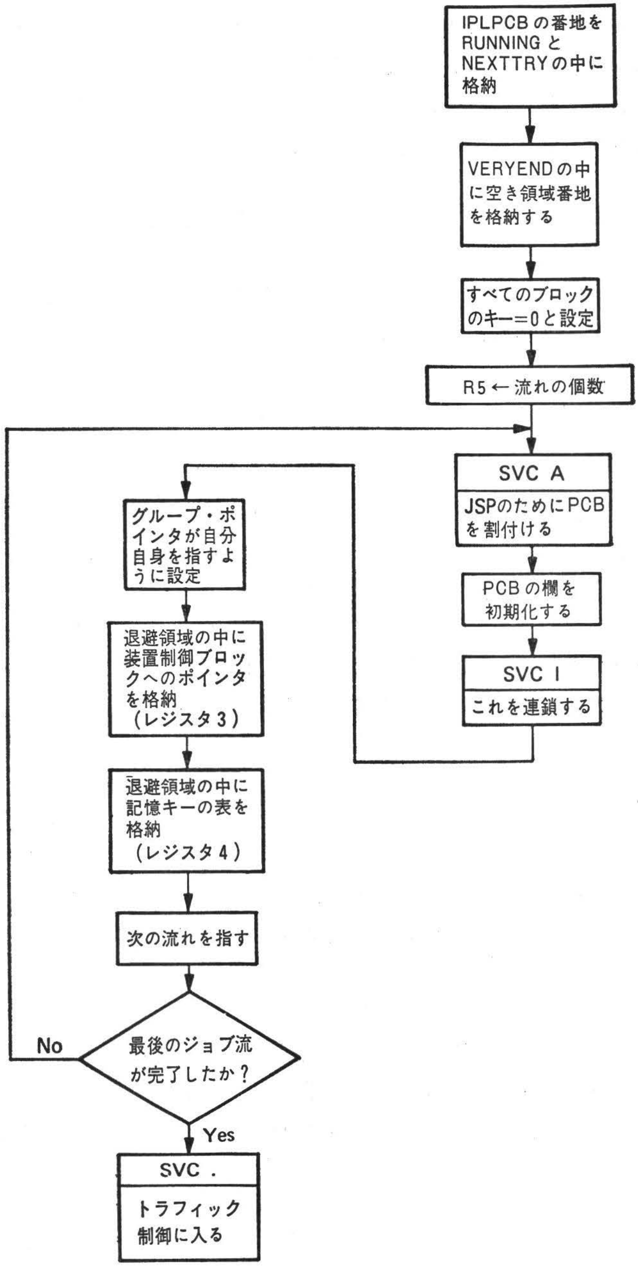


図 7.10 初期プログラム・ロード・ルーチン（IPLRTN）

利用者プログラムが使用する保護キーを納めている．これらのスーパバイザ・プロセスをすべて起動させると，初期化ルーチンは自分の PCB を変更して“停止”状態とし，ルーチン XPER を介して，トラフィック制御に移り，スーパバイザ・プロセスの実行を開始する．初期化ルーチンの流れ図は 図 7.10 に示されている．

スーパバイザ・プロセスは，実行し始めると，まず，前述したようにレジスタ3に納められた UCB へのポインタとレジスタ4の中の保護キーを引渡して，*IN および *OUT という名前のプロセスを作って起動する．ここでジョブの処理が開始される．

7.15　利用者プログラムとプロセス

利用者プログラムは，1つの利用者プロセスまたは幾つかのプロセスによって実行される．前述のように，これらのプロセスは USERPROG と呼ばれる1つのプロセスによって実行が開始される．しかし，それ以後にもっと多くのプロセスを作ることができる．システムには副プロセス創成の機能が備わっている．利用者プロセスは＊で始まる名前のプロセスを作ったり，＊で始まる名前のプロセスを破壊したりすることはできない．このようにして，スーパバイザ・プロセスは利用者プロセスから保護されている．

7.15.1　利用者プログラムの終了

XH および XQUE の2つのルーチンが，それぞれ，成功および失敗を通知する．通常，XH は利用者プログラムから呼出され，XQUE は，誤り条件を検出したとき利用者が呼出すシステム・ルーチンによって呼出される．いずれのルーチンもスーパバイザ・プロセスにメッセージを送り，閉塞状態となる．

7.15.2　利用者プログラムの実行

利用者プログラムは，“メッセージ送信”や“メッセージ受信”などの多くのオペレーティング・システムの基本機能を使用することができる．入力および出力は，それぞれ，入力プロセスからメッセージを受信したり，出力プロセスにメッセージを送信したりして実行することがきる．

きわめて簡単な利用者プログラムが 図 7.11 に示されている．このプログラムは単に USER PROGRAM ENTRY SUCCESSFULL というメッセージを印刷して終了する．図の最初には ＄JOB 制御カードも示されている．MSGONE

• $JOB,2K,READER=IN,PRINTER=OUT

番地	目的コード	番地1	番地2	文番号	原始文			註釈[1]
00000				1	USER	CSECT		
00000	05F0			2		BALR	15,0	番地指定の準備
00002				3		USING	*,15	
				4	*			
00002	4120 F00E		00010	5		LA	2,MSGONE	印刷機にメッセージを送る
00006	0AE2			6		SVC	C'S'	...
				7	*			
00008	4120 F0A6		000A8	8		LA	2,MSGTWO	印刷が完了したということを示す返答を待合せる
0000C	0AD9			9		SVC	C'R'	
				10	*			
0000E	0AC8			11		SVC	C'H'	ジョブを停止する
				12	*			
00010	D7D9C9D5E3C5D940			13	MSGONE	DC	CL8'PRINTER'	... プロセス名
00018	00000008			14		DC	F'8'	... メッセージ長
0001C	D7D9C9D500000024			15		DC	C 'PRIN',A(PRINT1)	... メッセージ本文
00024	E4E2C5D940D7D9D6			16	PRINT1	DC	CL132 'USER PROGRAM ENTRY SUCCESSFUL.'	印刷すべき本文
				17	*			
000A8				18	MSGTWO	DS	CL8	... プロセス名がここに返却される
000B0	00000004			19		DC	F'4'	... 返却用の領域の長さ
000B4				20		DS	CL4	... "印刷機"から返却される状態
				21		END		

[1] この註釈部は本来原始文の一部であるので原文のまま示すのが正しいが，理解を容易にするために翻訳しておいた(訳者)

図 7.11 利用者ジョブの例

の中で使用されている名前 PRINTER は，$JOB カードの PRINTER=OUT と一致しなければならない．

7.16　SVC の流れの軌跡

IPLRTN から開始した SVC 命令実行の順番と番地が 図 7.12 に示されている．この軌跡は，M.I.T. で開発された特別の IBM 360 シミュレータ SIM 360 を用いて求めたものである (Donovan, 1974)．簡単のために，軌跡の一部だけを示している．この軌跡と第 7.17 節のプログラム・リストとを用いて，システムの制御の流れを追うことが可能である．

```
MASSACHUSETTS INSTITUTE OF TECHNOLOGY                 SYSTEM / 360 SIMULATOR
PROGRAM TRACE LISTING.   0 CONDITIONS ENABLED.

LOC  TRACE TYPE          COUNT   TIME
---- ---------------     -----  ----------


0FA4 SVC INTERRUPT  A      78          85  PSW: 00 0 0 00C1  '01'B '00'B 0 000FA4

0610 SVC INTERRUPT  !     104         173  PSW: FF 0 0 005A  '01'B '00'B 0 000610

061C SVC INTERRUPT  P     139         293  PSW: FF 0 0 00D7  '01'B '00'B 0 00061C

069C SVC INTERRUPT  B     201         432  PSW: FF 0 0 00C2  '01'B '10'B 0 00069C

06A2 SVC INTERRUPT  V     239         554  PSW: FF 0 0 00E5  '01'B '10'B 0 0006A2

06B2 SVC INTERRUPT  ,     272         674  PSW: FF 0 0 006B  '01'B '10'B 0 0006B2

0FB0 SVC INTERRUPT  I     314         855  PSW: 00 0 0 00C9  '01'B '00'B 0 000FB0

0FA4 SVC INTERRUPT  A     365         990  PSW: 00 0 0 00C1  '01'B '10'B 0 000FA4

0610 SVC INTERRUPT  !     391       1,078  PSW: FF 0 0 005A  '01'B '00'B 0 000610

061C SVC INTERRUPT  P     426       1,198  PSW: FF 0 0 00D7  '01'B '00'B 0 00061C

069C SVC INTERRUPT  B     488       1,337  PSW: FF 0 0 00C2  '01'B '10'B 0 00069C

06A2 SVC INTERRUPT  V     526       1,459  PSW: FF 0 0 00E5  '01'B '10'B 0 0006A2

06B2 SVC INTERRUPT  ,     559       1,579  PSW: FF 0 0 006B  '01'B '10'B 0 0006B2

0FB0 SVC INTERRUPT  I     601       1,760  PSW: 00 0 0 00C9  '01'B '10'B 0 000FB0

0FE4 SVC INTERRUPT  .     651       1,895  PSW: 00 0 0 004B  '01'B '10'B 0 000FE4

11BC SVC INTERRUPT  P     699       2,004  PSW: FF 0 0 00D7  '01'B '00'B 0 0011BC

11C2 SVC INTERRUPT  E     732       2,123  PSW: FF 0 0 00C5  '01'B '00'B 0 0011C2
```

図 7.12 SVC の流れのトレースの一部（その1）

```
MASSACHUSETTS INSTITUTE OF TECHNOLOGY                SYSTEM / 360 SIMULATOR
PROGRAM TRACE LISTING.   1 CONDITIONS ENABLED.

LOC  TRACE TYPE        COUNT   TIME
---- ---------------- ----- ----------

0610 SVC INTERRUPT !     758      2,210  PSW: FF 0 0 005A  '01'B '00'B 0 000610

061C SVC INTERRUPT P     793      2,330  PSW: FF 0 0 00D7  '01'B '00'B 0 00061C

069C SVC INTERRUPT B     855      2,469  PSW: FF 0 0 00C2  '01'B '10'B 0 00069C

06A2 SVC INTERRUPT V     893      2,591  PSW: FF 0 0 00E5  '01'B '10'B 0 0006A2

06B2 SVC INTERRUPT ,     928      2,712  PSW: FF 0 0 006B  '01'B '00'B 0 0006B2

11CC SVC INTERRUPT V     970      2,864  PSW: FF 0 0 00E5  '01'B '00'B 0 0011CC

122E SVC INTERRUPT C    1020      3,034  PSW: FF 0 0 00C3  '01'B '00'B 0 00122E

078E SVC INTERRUPT N    1044      3,124  PSW: FF 0 0 00D5  '01'B '00'B 0 00078E

079A SVC INTERRUPT !    1079      3,246  PSW: FF 0 0 005A  '01'B '00'B 0 00079A

07A8 SVC INTERRUPT A    1113      3,372  PSW: FF 0 0 00C1  '01'B '00'B 0 0007A8

0610 SVC INTERRUPT !    1139      3,460  PSW: FF 0 0 005A  '01'B '00'B 0 000610

061C SVC INTERRUPT P    1174      3,580  PSW: FF 0 0 00D7  '01'B '00'B 0 00061C

069C SVC INTERRUPT B    1236      3,719  PSW: FF 0 0 00C2  '01'B '10'B 0 00069C

06A2 SVC INTERRUPT V    1274      3,840  PSW: FF 0 0 00E5  '01'B '10'B 0 0006A2

06B2 SVC INTERRUPT ,    1307      3,960  PSW: FF 0 0 006B  '01'B '10'B 0 0006B2

07BE SVC INTERRUPT I    1349      4,135  PSW: FF 0 0 00C9  '01'B '00'B 0 0007BE

07C0 SVC INTERRUPT ,    1387      4,257  PSW: FF 0 0 006B  '01'B '00'B 0 0007C0

1230 SVC INTERRUPT Y    1427      4,407  PSW: FF 0 0 00E8  '01'B '00'B 0 001230
```

図 7.12 SVC の流れのトレースの一部（その2）

```
MASSACHUSETTS INSTITUTE OF TECHNOLOGY              SYSTEM / 360 SIMULATOR
PROGRAM TRACE LISTING.   1 CONDITIONS ENABLED.

LOC  TRACE TYPE         COUNT   TIME
---- ---------------    -----  ----------


0A18 SVC INTERRUPT   N   1451      4,497  PSW: 00 0 0 00D5  '01'B '00'B 0 000A18

123A SVC INTERRUPT   C   1495      4,681  PSW: FF 0 0 00C3  '01'B '00'B 0 00123A

078E SVC INTERRUPT   N   1519      4,770  PSW: FF 0 0 00D5  '01'B '00'B 0 00078E

079A SVC INTERRUPT   !   1559      4,897  PSW: FF 0 0 005A  '01'B '00'B 0 00079A

07A8 SVC INTERRUPT   A   1593      5,023  PSW: FF 0 0 00C1  '01'B '00'B 0 0007A8

0610 SVC INTERRUPT   !   1619      5,111  PSW: FF 0 0 005A  '01'B '00'B 0 000610

061C SVC INTERRUPT   P   1654      5,231  PSW: FF 0 0 00D7  '01'B '00'B 0 00061C

069C SVC INTERRUPT   B   1716      5,369  PSW: FF 0 0 00C2  '01'B '10'B 0 00069C

06A2 SVC INTERRUPT   V   1754      5,491  PSW: FF 0 0 00E5  '01'B '10'B 0 0006A2

06B2 SVC INTERRUPT   ,   1787      5,611  PSW: FF 0 0 006B  '01'B '10'B 0 0006B2

07BE SVC INTERRUPT   I   1829      5,786  PSW: FF 0 0 00C9  '01'B '00'B 0 0007BE

07C0 SVC INTERRUPT   ,   1867      5,907  PSW: FF 0 0 006B  '01'B '00'B 0 0007C0

123C SVC INTERRUPT   Y   1907      6,058  PSW: FF 0 0 00E8  '01'B '00'B 0 00123C

0A18 SVC INTERRUPT   N   1931      6,147  PSW: 00 0 0 00D5  '01'B '00'B 0 000A18


NORMAL SIMULATOR TERMINATION FOR PROGRAM PROGRAM

           SIMULATED REAL TIME:       6,330.562

           SIMULATED CPU TIME:       6,330.562
```

図 7.12 SVCの流れのトレースの一部（その3）

7.17 例題オペレーティング・システムのプログラム・リスト

以下のページは例題オペレーティング・システムの実際のプログラム・リストを示している．この IBM 370 アセンブラ語のプログラムは，組込みの詳細を学ぼうとしたり，実際に例題オペレーティング・システムを実行しようとする読者のために用意されたものである．

このプログラムは試験ずみではあるが，このプログラムの主たる目的は一般的な構造を説明しようとすることにある．したがって，プログラムが正しいことを保証しない．本章末の演習問題では，このオペレーティング・システムの設計に変更を加えたり修正を行なったりする問題が示されている．

```
        SAMPLE OPERATING SYSTEM      VERSION 2.00

   LOC  OBJECT CODE      ADDR1 ADDR2 STMT   SOURCE STATEMENT

                                     2331+***********************************************************************
                                     2332+*                                                                     *
                                     2333+*  ****************************************************************   *
                                     2334+*  *                                                              *   *
                                     2335+*  *                   SAMPLE OPERATING SYSTEM                    *   *
                                     2336+*  *                        VERSION 2.00                          *   *
                                     2337+*  *                   DEVELOPED AT MIT, 1973                     *   *
                                     2338+*  *                                                              *   *
                                     2339+*  ****************************************************************   *
                                     2340+*                                                                     *
                                     2341+***********************************************************************


                                     2343+          PRINT ON,NODATA,GEN
 000000                              2344+PROGRAM   CSECT

 008000                              2346+CORESIZE  EQU   32768 BYTES OF CORE IN OBJECT MACHINE

 000000                              2348+          USING *,0 COMMUNICATIONS AREA

 000000 0000000000000F6E             2350+IPLPSW    DC    B'00000000',B'00000000',X'0000',X'00',AL3(IPLRTN)
 000008                              2351+IPLCCW1   DS    D .                         IPL CCW #1
 000010                              2352+IPLCCW2   DS    D .                         IPL CCW #2
 000018                              2353+EXTOLD    DS    D .                         EXTERNAL OLD PSW
 000020                              2354+SVCOLD    DS    D .                         SVC OLD PSW
 000028                              2355+PGMOLD    DS    D .                         PROGRAM INTERRUPT OLD PSW
 000030                              2356+MCHKOLD   DS    D .                         MACHINE CHECK OLD PSW
 000038                              2357+IOOLD     DS    D .                         I/O INTERRUPT OLD PSW
 000040                              2358+CSW       DS    D .                         CHANNEL STATUS WORD
 000048                              2359+CAW       DS    F .                         CHANNEL ADDRESS WORD
 00004C                              2360+UNUSED0   DS    F .
 000050 FFFFFFFF                     2361+TIMER     DC    F'-1' .                     TIMER
 000054 00000000                     2362+UNUSED1   DC    F'0'
 000058 000000000000027A             2363+EXTNEW    DC    B'00000000',B'00000000',X'0000',X'00',AL3(EXTHANDL)
 000060 00000000000002B2             2364+SVCNEW    DC    B'00000000',B'00000000',X'0000',X'00',AL3(SVCHANDL)
 000068 00000000000002B0             2365+PGMNEW    DC    B'00000000',B'00000000',X'0000',X'00',AL3(PGMHANDL)
 000070 0002000000000000             2366+MCHKNEW   DC    B'00000000',B'00000010',X'0000',X'00',AL3(0)
 000076 0000000000000EF4             2367+IONEW     DC    B'00000000',B'00000000',X'0000',X'00',AL3(IOHANDL)
 000080 000180                       2368+          ORG   *+X'100' SPACE OVER STAND ALONE DUMP AREA
 000180 00001648                     2369+FSBPTR    DC    A(VERYEND) .                FSB POINTER
 000184 0000000100000000             2370+FSBSEM    DC    F'1,0' .                    FSB SEMAPHORE
 00018C 0000000000000000             2371+MEMORY    DC    F'0,0' .                    MEMORY SEMAPHORE
 000194 0000000100000000             2372+CAWSEM    DC    F'1,0' .                    CAW SEMAPHORE

 00019C                              2374+TRAPSAVE  DS    16F .                       STORAGE FOR EXTERNAL INTERRUPTS
 0001DC                              2375+IOHSAVE   DS    16F .                       STORAGE FOR I/O INTERRUPTS

 00021C                              2377+SYSSEMSA  DS    CL84 .                      SYSTEM SEMAPHORE SAVE AREA

 000270                              2379+RUNNING   DS    A .                         RUNNING
 000274                              2380+NEXTTRY   DS    A .                         NEXTTRY
 000278                              2381+NEXTTRYM  DS    C,0H .                      NEXTTRY MODIFIED
```

```
           SAMPLE OPERATING SYSTEM     VERSION 2.00

  LOC  OBJECT CODE      ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                     2383+*****************************************************************
                                     2384+*                                                               *
                                     2385+*              EXTERNAL, PROGRAM, AND SVC INTERRUPT HANDLERS    *
                                     2386+*                                                               *
                                     2387+*****************************************************************
00027A                               2389+EXTHANDL EQU   * .                     EXTERNAL INTERRUPT HANDLER
00027A 900F 019C              0019C  2390+         STM   0,15,TRAPSAVE .         SAVE REGISTERS
00027E 0510                          2391+         BALR  1,0 .                   ESTABLISH ADDRESSING
000280                               2392+         USING *,1
000280 9580 001B        0001B        2393+         CLI   EXTOLD+3,X'80' .        SEE IF TIMER TRAP
000284 4770 1028              002A8  2394+         BNE   EXTHRET .               IF NOT, IGNORE
000288 58F0 0270              00270  2395+         L     15,RUNNING .            SET UP REGISTERS FOR TRAFFIC
000000                               2396+         USING PCB,15 .                CONTROLLER (XPER)
00028C 95FF F019        00019        2397+         CLI   PCBBLOKT,X'FF' .        IF BLOCKED, NO PROCESS IS
000290 4780 1028              002A8  2398+         BE    EXTHRET .                RUNNABLE, SO RETURN
000294 41E0 F04C              0004C  2399+         LA    14,PCBISA .             GET SAVE AREA
000000                               2400+         USING SA,14
000298 D207 E000 0018 00000 00018    2401+         MVC   SAPSW,EXTOLD .          AND STORE OLD STUFF INTO IT
00029E D23F E008 019C 00008 0019C    2402+         MVC   SAREGS,TRAPSAVE
0002A4 47F0 12EA              0056A  2403+         B     XPER .                  THEN GO TO TRAFFIC SCHEDULER
                                     2404+         DROP  14,15
0002A8 980F 019C              0019C  2405+EXTHRET  LM    0,15,TRAPSAVE .         TO IGNORE AN INTERRUPT, RELOAD
0002AC 8200 0018        00018        2406+         LPSW  EXTOLD .                AND TRANSFER BACK

0002B0                               2408+PGMHANDL EQU   * .                     PROGRAM INTERRUPT HANDLER
0002B0 0A6F                          2409+         SVC   C'?' .                  IN ANY CASE, AN ERROR
```

```
            SAMPLE OPERATING SYSTEM     VERSION 2.00

   LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                    2411+************************************************************************
                                    2412+*                                                                      *
                                    2413+*                         SVC INTERRUPT HANDLER                        *
                                    2414+*                                                                      *
                                    2415+*         FOR ALL ROUTINES ENTERED BY SVC INTERRUPT, THE               *
                                    2416+*         FOLLOWING REGISTERS CONTAIN THIS INFORMATION:                *
                                    2417+*                                                                      *
                                    2418+*         REGISTER  1 - BASE REGISTER FOR ROUTINE                      *
                                    2419+*         REGISTER  2 - POINTER TO ARGUMENT LIST (IF ANY)              *
                                    2420+*         REGISTER 14 - POINTER TO SAVEAREA USED FOR THIS SVC          *
                                    2421+*         REGISTER 15 - POINTER TO PCB PRESENTLY RUNNING               *
                                    2422+*                                                                      *
                                    2423+************************************************************************
0002B2                              2425+SVCHANDL EQU   * .                       SVC HANDLER
0002B2 900F 019C              0019C 2426+         STM   0,15,TRAPSAVE .           SAVE REGISTERS
0002B6 0590                         2427+         BALR  9,0 .                     ESTABLISH ADDRESSING
0002B8                              2428+         USING *,9
0002B8 98AE 905C              00314 2429+         LM    10,14,SVCCONST .          INITIALIZE REGISTERS
0002BC 43A0 0023              00023 2430+         IC    10,SVCOLD+3 .             GET SVC CODE
0002C0 43AA 9070              00328 2431+         IC    10,SVCHTABL(10) .         TRANSLATE INTO TABLE OFFSET
0002C4 41AA 9170              00428 2432+         LA    10,SVCRTN(10) .           REG 10 -> THE CORRECT PSW
0002C8 9500 A002        00002       2433+         CLI   2(10),X'00' .             IS THIS CALL PROTECTED?
0002CC 4780 904A              00302 2434+         BE    SVCHPROT .                THEN SEE IF WE CAN CALL IT
0002D0 58F0 0270              00270 2435+SVCOK    L     15,RUNNING .              GET PCB POINTER
000000                              2436+         USING PCB,15
0002D4 9500 A003        00003       2437+         CLI   3(10),X'00' .             IS IT A SYSTEM SAVEAREA?
0002D8 4780 9026              002DE 2438+         BE    SYSSEM .                  DON'T USE REG 14 AS PCB PTR
0002DC 18EF                         2439+         LR    14,15 .                   ELSE, SET UP PCB POINTER
0002DE 43BA 0003              00003 2440+SYSSEM   IC    11,3(10) .                GET POINTER TO SAVE AREA OFFSET
0002E2 5AEB 9210              004C8 2441+         A     14,SVCSAVE(11) .          REG 14 -> SAVE AREA
0002E6 954B 0023        00023       2442+         CLI   SVCOLD+3,C'.' .           ARE WE CALLING XPER?
0002EA 4780 9042              002FA 2443+         BE    SVCXPER .                 IF SO, DON'T SAVE RETURN STATUS
000000                              2444+         USING SA,14
0002EE D207 E000 0020 00000 00020   2445+         MVC   SAPSW,SVCOLD .            SAVE PSW
0002F4 D23F E008 019C 00008 0019C   2446+         MVC   SAREGS,TRAPSAVE .         SAVE REGISTERS
0002FA 581A 0004              00004 2447+SVCXPER  L     1,4(10) .                 MAKE ADDRESSING EASY WITHIN
0002FE 8200 A000        00000       2448+         LPSW  0(10) .                    ROUTINE, AND GO THERE
000302 58C0 0020              00020 2449+SVCHPROT L     12,SVCOLD .               GET PROTECTION KEY
000306 14CD                         2450+         NR    12,13 .                   IS IT A USER?
000308 4780 9018              002D0 2451+         BZ    SVCOK .                   IF NO, THAT'S FINE
00030C 41A0 91F8              004B0 2452+         LA    10,SVCRTN+136 .           ELSE SET UP CALL TO XQUE
000310 47F0 9018              002D0 2453+         B     SVCOK .
                                    2454+         DROP  9
000314 0000000000000000             2455+SVCCONST DC    3F'0',X'00F00000',F'0'

000328 8484848484848484             2457+SVCHTABL DC    256X'84' .                TABLE OF PSW OFFSETS
000428 0003FF                       2458+         ORG   SVCHTABL+C'P'
0003FF 00                           2459+         DC    AL1(0)
000400 00040D                       2460+         ORG   SVCHTABL+C'Y'
00040D 08                           2461+         DC    AL1(8)
00040E 000382                       2462+         ORG   SVCHTABL+C'!'
000382 10                           2463+         DC    AL1(16)
000383 000393                       2464+         ORG   SVCHTABL+C','
000393 18                           2465+         DC    AL1(24)
```

```
         SAMPLE OPERATING SYSTEM        VERSION 2.00

  LOC  OBJECT CODE     ADDR1 ADDR2   STMT    SOURCE STATEMENT

000394 0003EA                        2466+           ORG    SVCHTABL+C'B'
0003EA 20                            2467+           DC     AL1(32)
0003EB 0003E9                        2468+           ORG    SVCHTABL+C'A'
0003E9 28                            2469+           DC     AL1(40)
0003EA 0003EE                        2470+           ORG    SVCHTABL+C'F'
0003EE 30                            2471+           DC     AL1(48)
0003EF 0003F1                        2472+           ORG    SVCHTABL+C'I'
0003F1 38                            2473+           DC     AL1(56)
0003F2 0003F9                        2474+           ORG    SVCHTABL+C'J'
0003F9 40                            2475+           DC     AL1(64)
0003FA 000373                        2476+           ORG    SVCHTABL+C'.'
000373 48                            2477+           DC     AL1(72)
000374 000401                        2478+           ORG    SVCHTABL+C'R'
000401 50                            2479+           DC     AL1(80)
000402 00040A                        2480+           ORG    SVGHTABL+C'S'
00040A 58                            2481+           DC     AL1(88)
00040B 0003EB                        2482+           ORG    SVCHTABL+C'C'
0003EB 60                            2483+           DC     AL1(96)
0003EC 0003FD                        2484+           ORG    SVCHTABL+C'N'
0003FD 68                            2485+           DC     AL1(104)
0003FE 000410                        2486+           ORG    SVCHTABL+C'Y'
000410 70                            2487+           DC     AL1(112)
000411 000411                        2488+           ORG    SVCHTABL+C'Z'
000411 78                            2489+           DC     AL1(120)
000412 0003EC                        2490+           ORG    SVCHTABL+C'D'
0003EC 80                            2491+           DC     AL1(128)
0003ED 000397                        2492+           ORG    SVCHTABL+C'?'
000397 88                            2493+           DC     AL1(136)
000398 0003F0                        2494+           ORG    SVCHTABL+C'H'
0003F0 90                            2495+           DC     AL1(144)
0003F1 0003ED                        2496+           ORG    SVCHTABL+C'E'
0003ED 98                            2497+           DC     AL1(152)
0003EE 000428                        2498+           ORG    SVCHTABL+256

000428                               2500+SVCRTN     DS     0D .              THE PSWS
                                     2501+*             IN THE FOLLOWING PSWS, THE THIRD BYTE INDICATES     *
                                     2502+*             WHETHER THE SVC IS RESTRICTED:                      *
                                     2503+*                       X'00' -> OPERATING SYSTEM ONLY            *
                                     2504+*                       X'FF' -> AVAILABLE TO USER ALSO           *
                                     2505+*                                                                 *
                                     2506+*             THE FOURTH BYTE INDICATES WHICH SAVE AREA TO USE;   *
                                     2507+*             SVCSAVE BELOW SHOWS THE CODE VALUES.                *
000428 00000000000004EE              2508+           DC     B'00000000',B'00000000',X'0000',X'00',AL3(XP)
000430 0000000000000534              2509+           DC     B'00000000',B'00000000',X'0000',X'00',AL3(XV)
000438 00000004000005C0              2510+           DC     B'00000000',B'00000000',X'0004',X'00',AL3(XEXC)
000440 00000004000005D2              2511+           DC     B'00000000',B'00000000',X'0004',X'00',AL3(XCOM)
000448 0000000400000744              2512+           DC     B'00000000',B'00000000',X'0004',X'00',AL3(XB)
000450 FF00000C00000600              2513+           DC     B'11111111',B'00000000',X'000C',X'00',AL3(XA)
000458 FF00000C000006B6              2514+           DC     B'11111111',B'00000000',X'000C',X'00',AL3(XF)
000460 000000040000087A              2515+           DC     B'00000000',B'00000000',X'0004',X'00',AL3(XI)
000468 00000004000008A6              2516+           DC     B'00000000',B'00000000',X'0004',X'00',AL3(XJ)
000470 000000040000056A              2517+           DC     B'00000000',B'00000000',X'0004',X'00',AL3(XPER)
000478 FF00FF08000008EC              2518+           DC     B'11111111',B'00000000',X'FF08',X'00',AL3(XR)
000480 FF00FF0800000978              2519+           DC     B'11111111',B'00000000',X'FF08',X'00',AL3(XS)
000488 FF00FF0800000780              2520+           DC     B'11111111',B'00000000',X'FF08',X'00',AL3(XC)
```

```
          SAMPLE OPERATING SYSTEM     VERSION 2.00

  LOC  OBJECT CODE      ADDR1 ADDR2  STMT    SOURCE STATEMENT

000490 0000FF04000008CA              2521+         DC    B'00000000',B'00000000',X'FF04',X'00',AL3(XN)
000498 0000FF0800000A0A              2522+         DC    B'00000000',B'00000000',X'FF08',X'00',AL3(XY)
0004A0 FF00FF0800000A42              2523+         DC    B'11111111',B'00000000',X'FF08',X'00',AL3(XZ)
0004A8 FF00FF08000007C6              2524+         DC    B'11111111',B'00000000',X'FF08',X'00',AL3(XD)
0004B0 0000FF0400000A8E              2525+         DC    B'00000000',B'00000000',X'FF04',X'00',AL3(XQUE)
0004B8 FF00FF0800000842              2526+         DC    B'11111111',B'00000000',X'FF08',X'00',AL3(XH)
0004C0 FF00000C00000608              2527+         DC    B'11111111',B'00000000',X'000C',X'00',AL3(XAUTO)

0004C8                               2529+SVCSAVE  DS    0F .                           THE SAVE AREA OFFSETS
0004C8 0000021C                      2530+         DC    A(SYSSEMSA) .                  CODE 00 -> SYSSEMSA
0004CC 0000004C                      2531+         DC    A(PCBISA-PCB) .                CODE 04 -> INTERRUPT SAVE AREA
0004D0 000000A0                      2532+         DC    A(PCBFSA-PCB) .                CODE 08 -> FAULT SAVE AREA
0004D4 000000F4                      2533+         DC    A(PCBMSA-PCB) .                CODE 0C -> MEMORY SAVE AREA

                                     2535+***********************************************************************
                                     2536+*                                                                     *
                                     2537+* RETURN SEQUENCE FOR REQUEST DRIVEN ROUTINES AND TRAFFIC CONTROLLER  *
                                     2538+*                                                                     *
                                     2539+***********************************************************************

0004D8                               2541+         DS    0D
0004D8 00000000000004E0              2542+RETURN   DC    B'00000000',B'00000000',X'0000',X'00',AL3(RETURNR)

0004E0                               2544+RETURNR  EQU   * .                            RETURN ROUTINE FOR SVC'S & XPER
0004E0 D207 0020 E000 00020 00000    2545+         MVC   SVCOLD,SAPSW .                 SAVE PSW IN A SAFE PLACE
0004E6 980F E008            00008    2546+         LM    0,15,SAREGS .                  RELOAD REGISTERS
0004EA 8200 0020      00020          2547+         LPSW  SVCOLD .                       AND RETURN
```

```
          SAMPLE OPERATING SYSTEM     VERSION 2.00

   LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                    2549+***********************************************************************
                                    2550+*                                                                      *
                                    2551+*                          REQUEST DRIVEN ROUTINES                     *
                                    2552+*                                                                      *
                                    2553+***********************************************************************

                                    2555+***********************************************************************
                                    2556+*                                                                      *
                                    2557+*                              XP ROUTINE                              *
                                    2558+*                                                                      *
                                    2559+*      FUNCTION: TO IMPLEMENT "P" PRIMITIVE FOR SEMAPHORES             *
                                    2560+*     DATABASES: UPON ENTRY, REGISTER 2 CONTAINS ADDRESS SM            *
                                    2561+*                 SM        DS 0D    SEMAPHORE DEFINITION              *
                                    2562+*                 SMVAL     DS F     VALUE                             *
                                    2563+*                 SMPTR     DS A     POINTER TO FIRST WAITER           *
                                    2564+*  ROUTINES USED: XPER                                                 *
                                    2565+*     PROCEDURE: SUBTRACT ONE FROM SMVAL; IF NON-NEGATIVE, RETURN.     *
                                    2566+*                IF NEGATIVE, PLACE RUNNING PROCESS AT END OF LIST     *
                                    2567+*                OF PROCESSES WAITING ON SM. BLOCK CALLING PROCESS;    *
                                    2568+*                ENTER TRAFFIC CONTROLLER.                             *
                                    2569+*   ERROR CHECKS: NONE                                                 *
                                    2570+*     INTERRUPTS: OFF                                                  *
                                    2571+*    USER ACCESS: NO                                                   *
                                    2572+*                                                                      *
                                    2573+***********************************************************************
 0004EE                             2575+XP       EQU   * .                            THE XP ROUTINE
 0004EE                             2576+         USING *,1
 000000                             2577+         USING SM,2 .                         ARGUMENT IS A SEMAPHORE
 0004EE 5830 2000            00000  2578+         L     3,SMVAL .                      GET THE VALUE
 0004F2 0630                        2579+         BCTR  3,0 .                          SUBTRACT ONE
 0004F4 5030 2000            00000  2580+         ST    3,SMVAL .                      AND STORE IT BACK
 0004F8 1233                        2581+         LTR   3,3 .                          SET CONDITION CODE
 0004FA 4740 1014            00502  2582+         BM    XPWAIT .                       IF IT'S NEGATIVE, MUST WAIT
 0004FE 8200 04D8      004D8        2583+         LPSW  RETURN .                       ELSE RETURN NOW
 000502 4140 2004            00004  2584+XPWAIT   LA    4,SMPTR .                      START GOING DOWN
 000506 5850 2004            00004  2585+         L     5,SMPTR .                       CHAIN OF POINTERS
                                    2586+         DROP  15
 000000                             2587+         USING PCB,5
 00050A 1255                        2588+XPLOOP   LTR   5,5 .                          IF REACHED END
 00050C 4780 102E            0051C  2589+         BZ    XPTHEN .                       ADD OUR PCB ON. ELSE,
 000510 4140 5030            00030  2590+         LA    4,PCBNSW .                     INCREMENT POINTERS
 000514 5850 5030            00030  2591+         L     5,PCBNSW
 000518 47F0 101C            0050A  2592+         B     XPLOOP .                       AND TRY AGAIN
                                    2593+         DROP  5
 000000                             2594+         USING PCB,15
 00051C D203 4000 0270 00000 00270  2595+XPTHEN   MVC   0(4,4),RUNNING .               WE'RE AT THE END
 000522 5050 F030            00030  2596+         ST    5,PCBNSW .                     STORE NULL POINTER
 000526 92FF F019      00019        2597+         MVI   PCBBLOKT,X'FF' .               AND WE'RE BLOCKED
 00052A D253 F04C 021C 0004C 0021C  2598+         MVC   PCBISA,SYSSEMSA .              SWITCH SAVE AREAS
 000530 47F0 107C            0056A  2599+         B     XPER .                         SO RUN SOMEONE ELSE
                                    2600+         DROP  2
```

```
        SAMPLE OPERATING SYSTEM     VERSION 2.00

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT
                                   2602+***********************************************************************
                                   2603+*                                                                     *
                                   2604+*                              XV ROUTINE                             *
                                   2605+*                                                                     *
                                   2606+*      FUNCTION: TO IMPLEMENT "V" PRIMITIVE FOR SEMAPHORES            *
                                   2607+*     DATABASES: UPON ENTRY, REGISTER 2 CONTAINS ADDRESS SM           *
                                   2608+*                  SM          DS OD   SEMAPHORE DEFINITION          *
                                   2609+*                  SMVAL       DS F    VALUE                         *
                                   2610+*                  SMPTR       DS A    POINTER TO FIRST WAITER       *
                                   2611+*  ROUTINES USED: NONE                                                *
                                   2612+*     PROCEDURE: ADD ONE TO SMVAL; IF > ZERO, RETURN. IF ZERO OR     *
                                   2613+*                LESS, REMOVE FIRST PROCESS FROM WAITER CHAIN;       *
                                   2614+*                UNBLOCK IT; IF NEXTTRYM NOT SET, SET IT AND SET     *
                                   2615+*                NEXTTRY TO THAT PROCESS; RETURN; IF NEXTTRYM SET,   *
                                   2616+*                RETURN.                                             *
                                   2617+*   ERROR CHECKS: NONE                                                *
                                   2618+*     INTERRUPTS: OFF                                                 *
                                   2619+*    USER ACCESS: NO                                                  *
                                   2620+*                                                                     *
                                   2621+***********************************************************************
000534                             2623+XV        EQU   * .                            THE XV ROUTINE
000534                             2624+          USING *,1
000000                             2625+          USING SM,2 .                         ARGUMENT IS A SEMAPHORE
000534 5830 2000            00000  2626+          L     3,SMVAL .                      GET THE VALUE
000538 5A30 18F4            00E28  2627+          A     3,=F'1' .                      ADD ONE
00053C 5030 2000            00000  2628+          ST    3,SMVAL .                      AND STORE IT BACK
0005 0 47D0 1014            00548  2629+          BNP   XVWAKEUP .                     IF <=0, SOMEONE'S WAITING
000544 8200 04D8      004D8        2630+          LPSW  RETURN .                       ELSE RETURN
000548 5840 2004            00004  2631+XVWAKEUP  L     4,SMPTR .                      GET THE FIRST OF THE GUYS
                                   2632+          DROP  15
000000                             2633+          USING PCB,4
00054C D203 2004 4030 00004 00030  2634+          MVC   SMPTR,PCBNSW .                 REMEMBER THE REST
000552 9200 4019      00019        2635+          MVI   PCBBLOKT,X'00' .               WE'RE NO LONGER BLOCKING HIM
000556 95FF 0278      00278        2636+          CLI   NEXTTRYM,X'FF' .               IS NEXT TRY MODIFIED?
00055A 4780 1032            00566  2637+          BE    XVRET .                        IF SO, WELL OK
00055E 5040 0274            00274  2638+          ST    4,NEXTTRY .                    ELSE MODIFY NEXTTRY
000562 92FF 0278      00278        2639+          MVI   NEXTTRYM,X'FF' .               AND SAY SO
000566 8200 04D8      004D8        2640+XVRET     LPSW  RETURN .                       GET BACK
                                   2641+          DROP  2,4
```

```
         SAMPLE OPERATING SYSTEM      VERSION 2.00

    LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                     2643+***********************************************************************
                                     2644+*                                                                     *
                                     2645+*                        XPER ROUTINE (TRAFFIC CONTROLLER)            *
                                     2646+*                                                                     *
                                     2647+*        FUNCTION: TO IMPLEMENT MULTIPROGRAMING                       *
                                     2648+*       DATABASES: NONE                                               *
                                     2649+*   ROUTINES USED: NONE                                               *
                                     2650+*       PROCEDURE: STARTING WITH NEXTTRY, SEARCH FOR PROCESS ON ALL   *
                                     2651+*                  PCB CHAIN NOT BLOCKED OR STOPPED; IF FOUND, USE AS *
                                     2652+*                  NEW RUNNING, FOR 50 MS OF TIME AND RETURN. ELSE,   *
                                     2653+*                  ENTER WAIT STATE WITH INTERRUPTS ON, AND TRY TO    *
                                     2654+*                  SCHEDULE AGAIN AFTER INTERRUPT; RETURN.            *
                                     2655+*    ERROR CHECKS: NONE                                               *
                                     2656+*      INTERRUPTS: OFF                                                *
                                     2657+*     USER ACCESS: NO                                                 *
                                     2658+*                                                                     *
                                     2659+***********************************************************************
  00056A                             2661+XPER     EQU   * .                          ROUTINE XPER: TRAFFIC SCHEDULER
  00056A 8000 0078      00078        2662+         SSM   IONEW .                      MASK OFF INTERRUPTS
  00056E 0510                        2663+         BALR  1,0
  000570                             2664+         USING *,1
  000570 58A0 0274            00274  2665+         L     10,NEXTTRY .                 START LOOKING AT NEXTTRY
  000574 18BA                        2666+         LR    11,10 .                      REMEMBER WHICH THAT WAS.
  000000                             2667+         USING PCB,10
  000576 95FF A019      00019        2668+GWLOOP   CLI   PCBBLOKT,X'FF' .             IF IT'S BLOCKED,
  00057A 4780 1016            00586  2669+         BE    GWINC .                      IGNORE
  00057E 95FF A018      00018        2670+         CLI   PCBSTOPT,X'FF' .             ELSE, IF IT'S NOT STOPPED,
  000582 4770 1030            005A0  2671+         BNE   GWRUN .                      WE CAN RUN IT
  000586 58A0 A010            00010  2672+GWINC    L     10,PCBNPALL .                ELSE, GO TO THE NEXT
  00058A 19AB                        2673+         CR    10,11 .                      IF WE'VE SEEN ALL, QUIT
  00058C 4770 1006            00576  2674+         BNE   GWLOOP .                     ELSE TRY AGAIN
  000590 8200 1028      00598        2675+         LPSW  IDLE .                       SIT AND WAIT
  000598                             2676+         DS    0D
  000598 FE0200000000056A            2677+IDLE     DC    B'11111110',B'00000010',X'0000',X'00',AL3(XPER)

  0005A0 D203 0274 A010 00274 00010  2679+GWRUN    MVC   NEXTTRY,PCBNPALL .           GET A NEW NEXTTRY
  0005A6 9200 0278      00278        2680+         MVI   NEXTTRYM,X'00' .             NOT MODIFIED
  0005AA 50A0 0270            00270  2681+         ST    10,RUNNING .                 GET A NEW RUNNING
  0005AE 41E0 A04C            0004C  2682+         LA    14,PCBISA
  0005B2 D203 0050 104C 00050 005BC  2683+         MVC   TIMER,QUANTUM .              INTERRUPT AFTER 50 MS
  0005B8 8200 04D8      004D8        2684+         LPSW  RETURN .                     AND GO TO RETURNR
  0005BC 00000F00                    2685+QUANTUM  DC    X'00000F00' .                QUANTUM OF TIME
                                     2686+         DROP  10
  000000                             2687+         USING PCB,15
```

```
              SAMPLE OPERATING SYSTEM       VERSION 2.00

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT
                                   2689+***********************************************************************
                                   2690+*                                                                     *
                                   2691+*                              XEXC ROUTINE                           *
                                   2692+*                                                                     *
                                   2693+*       FUNCTION: TO ENTER SMC SECTION                                *
                                   2694+*      DATABASES: NONE                                                *
                                   2695+*  ROUTINES USED: NONE                                                *
                                   2696+*      PROCEDURE: INCREMENT SMC BYTE IN PCB BY ONE; RETURN.           *
                                   2697+*   ERROR CHECKS: NONE                                                *
                                   2658+*     INTERRUPTS: OFF                                                 *
                                   2699+*    USER ACCESS: NO                                                  *
                                   2700+*                                                                     *
                                   2701+***********************************************************************

0005C0                             2703+XEXC     EQU   *.                        ROUTINE XEXC: ENTER SMC SECTION
0005C0                             2704+         USING *,1
0005C0 1B88                        2705+         SR    8,8
0005C2 4380 F01A            0001A  2706+         IC    8,PCBINSMC
0005C6 4188 0001            00001  2707+         LA    8,1(8) .                  ADD ONE TO IN SMC BYTE
0005CA 4280 F01A            0001A  2708+         STC   8,PCBINSMC
0005CE 8200 04D8      004D8        2709+         LPSW  RETURN .                  AND LEAVE

                                   2711+***********************************************************************
                                   2712+*                                                                     *
                                   2713+*                              XCOM ROUTINE                           *
                                   2714+*                                                                     *
                                   2715+*       FUNCTION: TO LEAVE SMC SECTION                                *
                                   2716+*      DATABASES: NONE                                                *
                                   2717+*  ROUTINES USED: XP, XV                                              *
                                   2718+*      PROCEDURE: DECREMENT SMC BYTE IN PCB BY ONE; IF NOT ZERO,      *
                                   2719+*                 RETURN. ELSE, CHECK FOR STOP WAITING; IF STOP       *
                                   2720+*                 WAITING, ALLOW STOP AND BLOCK SELF; RETURN. IF NO   *
                                   2721+*                 STOP WAITING, RETURN.                               *
                                   2722+*   ERROR CHECKS: NONE                                                *
                                   2723+*     INTERRUPTS: OFF                                                 *
                                   2724+*    USER ACCESS: NO                                                  *
                                   2725+*                                                                     *
                                   2726+***********************************************************************

0005D2                             2728+XCOM     EQU   * .                       ROUTINE XCOM: LEAVE SMC
0005D2                             2729+         USING *,1
0005D2 1B88                        2730+         SR    8,8
0005D4 4380 F01A            0001A  2731+         IC    8,PCBINSMC
0005D8 0680                        2732+         BCTR  8,0 .                     SUBTRACT ONE FROM IN SMC BYTE
0005DA 4280 F01A            0001A  2733+         STC   8,PCBINSMC
0005DE 1288                        2734+         LTR   8,8 .                     IS IT ZERO?
0005E0 4770 102A            005FC  2735+         BNZ   XCOMRET .                 NO, THEN GET BACK. OTHERWISE,
0005E4 9500 F01B      0001B        2736+         CLI   PCBSW,X'00' .             IS STOP WAITING?
0005E8 4780 102A            005FC  2737+         BE    XCOMRET .                 IF NOT, RETURN
0005EC 9200 F01B      0001B        2738+         MVI   PCBSW,X'00' .             STOPS NOT WAITING AFTER THIS
0005F0 4120 F034            00034  2739+         LA    2,PCBSRS .                WE'LL "V" THE STOPPER,
0005F4 0AE5                        2740+         SVC   C'V'
0005F6 4120 F03C            0003C  2741+         LA    2,PCBSES .                AND "P" THE STOPPEE.
0005FA 0AD7                        2742+         SVC   C'P'
0005FC 8200 04D8      004D8        2743+XCOMRET  LPSW  RETURN .                  AND HERE (IF EVER) WE RETURN
```

```
        SAMPLE OPERATING SYSTEM      VERSION 2.00

LOC    OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                   2745+*********************************************************************
                                   2746+*                                                                   *
                                   2747+*                              XA ROUTINE                           *
                                   2748+*                           XAUTO ROUTINE                           *
                                   2749+*                                                                   *
                                   2750+*      FUNCTION: TO ALLOCATE MEMORY                                 *
                                   2751+*     DATABASES: UPON ENTRY, REGISTER 2 CONTAINS ADDRESS XAX:       *
                                   2752+*                  XAX       DS 0D                                  *
                                   2753+*                  XAXSIZE   DS F    SIZE OF BLOCK TO BE ALLOCATED  *
                                   2754+*                  XAXADDR   DS A    ADDRESS OF FIRST BYTE OF BLOCK *
                                   2755+*                  XAXALGN   DS F    ALIGNMENT OF BLOCK             *
                                   2756+* ROUTINES USED: XEXC, XCOM, XP, XV, XB                             *
                                   2757+*     PROCEDURE: LOCK FSB SEMAPHORE; SEARCH FREE STORAGE FOR LARGE  *
                                   2758+*                ENOUGH MEMORY BLOCK; ALIGN BOUNDARY; USE XB TO     *
                                   2759+*                CHAIN ANY LEFTOVER BLOCKS TO FREE STORAGE LIST;    *
                                   2760+*                PLACE ADDRESS OF ALLOCATED BLOCK IN XAXADDR; UNLOCK *
                                   2761+*                FSB SEMAPHORE; RETURN. IF CAN'T SATISFY REQUEST,   *
                                   2762+*                UNLOCK FSB SEMAPHORE, APPLY XP ROUTINE TO MEMORY   *
                                   2763+*                SEMAPHORE, BLOCKING PROCESS RUNNING UNTIL MEMORY   *
                                   2764+*                FREED; THEN UNBLOCK; TRY TO SATISFY REQUEST AGAIN. *
                                   2765+*  ERROR CHECKS: NONE                                               *
                                   2766+*    INTERRUPTS: ON                                                 *
                                   2767+*   USER ACCESS: NO                                                 *
                                   2768+*                                                                   *
                                   2769+*********************************************************************
000600                             2771+XA       EQU   * .                     THE XA ROUTINE, TO ALLOCATE
000600                             2772+         USING *,1
000600 4100 0001            00001  2773+         LA    0,1 .                   SET REGISTER ZERO TO ONE TO
000604 47F0 100E            0060E  2774+         B     XACOM .                  INDICATE C'A' CALL
000608                             2775+XAUTO    EQU   * .                     AUTO STORAGE ENTRY POINT
000608                             2776+         USING *,1
000608 1B00                        2777+         SR    0,0 .                   REG0=0 INDICATES C'E' CALL
00060A 5810 1824            00E2C  2778+         L     1,=A(XA) .              RESET BASE REGISTER PROPERLY
000600                             2779+         USING XA,1
00060E 0A5A                        2780+XACOM    SVC   C'!' .                  ENTER SMC
000610 1872                        2781+         LR    7,2
000000                             2782+         USING XAX,7 .                 ARGUMENT LIST
000612 5860 7000            00000  2783+         L     6,XAXSIZE .             GET THE SIZE REQUESTED
000616 4120 0184            00184  2784+XATOP    LA    2,FSBSEM .              LOCK THE FSB SEMAPHORE
00061A 0AD7                        2785+         SVC   C'P' .
00061C 4150 0180            00180  2786+         LA    5,FSBPTR .              START LOOKING DOWN
000620 5840 0180            00180  2787+         L     4,FSBPTR .               THE FREE STORAGE LIST
000624 5880 7008            00008  2788+         L     8,XAXALGN .             WE WOULD HAVE TO START AT WITH
000628 0680                        2789+         BCTR  8,0 .                    THIS CONSTANT TO FIND ALIGNMENT
000000                             2790+         USING FSB,4
00062A 1244                        2791+XALOOP   LTR   4,4 .                   IF AT THE END,
00062C 4780 1056            00656  2792+         BZ    XAWAIT .                 WAIT UNTIL A "FREE" OP
000630 18D4                        2793+         LR    13,4 .                  FIND THE LOCATION
000632 06D0                        2794+         BCTR  13,0 .                   IN THIS BLOCK WITH THIS
000634 16D8                        2795+         OR    13,8 .                  ALIGNMENT
000636 41DD 0001            00001  2796+         LA    13,1(13) .              THAT'S IT
00063A 189D                        2797+         LR    9,13 .                  AND NOW GET IN REG 9
00063C 1B94                        2798+         SR    9,4 .                    WHAT IS WASTED AT THE FRONT
00063E 5830 4004            00004  2799+         L     3,FSBSIZE .             GET SIZE MINUS WASTE AT
```

```
              SAMPLE OPERATING SYSTEM        VERSION 2.00

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT    SOURCE STATEMENT

000642 1B39                        2800+           SR    3,9 .                            FRONT, LEAVING EFFECTIVE SIZE
000644 1963                        2801+           CR    6,3 .                           IS IT ENOUGH?
000646 47D0 1062             00662 2802+           BNP   XAFOUND .                       EUREKA!
00064A 4150 4000             00000 2803+           LA    5,PSBNEXT .                     OH WELL, GET THE NEXT FREE
00064E 5840 4000             00000 2804+           L     4,PSBNEXT .                      STORAGE BLOCK ON THE CHAIN
000652 47F0 102A             0062A 2805+           B     XALOOP .                        BETTER LUCK NEXT TIME
000656 0AE5                        2806+XAWAIT     SVC   C'V' .                          NEED TO WAIT
000658 4120 018C             0018C 2807+           LA    2,MEMORY .                      SO WE LET OTHER PEOPLE GET IN
00065C 0AD7                        2808+           SVC   C'P' .                          SO THEY'LL WAKE US UP
00065E 47F0 1016             00616 2809+           B     XATOP .                         AND THEN WE'LL TRY AGAIN
000662 50D0 7004             00004 2810+XAFOUND    ST    13,XAXADDR .                    WE'VE NOW GOT THE ADDRESS
000666 D203 5000 4000 00000 00000 2811+           MVC   0(4,5),PSBNEXT .                UNLINK THE BLOCK OUT
00066C 58C0 4004             00004 2812+           L     12,PSBSIZE .                    GET THE WHOLE BLOCK SIZE
000670 4120 E048             00048 2813+           LA    2,SATEMP .                      START MAKING UP ARG LISTS
000000                             2814+           USING XBX,2 .                         FOR THE XB RTN
000674 18AD                        2815+           LR    10,13 .                         THE STARTING LOCATION
000676 1BA4                        2816+           SR    10,4 .                          MINUS THE START OF THE BLOCK
000678 4780 1086             00686 2817+           BZ    XANF .                          IF NONE WASTED AT FRONT, SKIP
00067C 5040 2004             00004 2818+           ST    4,XBXADDR .                     ELSE FREE, STARTING THERE
000680 50A0 2000             00000 2819+           ST    10,XBXSIZE .                     UP TO THE BEGINNING OF THE
000684 0AC2                        2820+           SVC   C'B' .                           ALLOCATION; INSERT IT IN CHAIN
000686 18BD                        2821+XANF       LR    11,13 .                         THE STARTING ADDR PLUS THE SIZE
000688 1AB6                        2822+           AR    11,6 .                           GIVES THE FIRST UNUSED ADDR
00068A 1BCA                        2823+           SR    12,10 .                         MINUS THE WASTE AT FRONT,
00068C 1BC6                        2824+           SR    12,6 .                          MINUS THE PART ALLOCATED. IF
00068E 4780 109C             0069C 2825+           BZ    XARETURN .                      NONE LEFT OVER, GOOD
000692 50B0 2004             00004 2826+           ST    11,XBXADDR .                    ELSE STORE ADDRESS AND
000696 50C0 2000             00000 2827+           ST    12,XBXSIZE .                    SIZE, AND LINK ONTO
00069A 0AC2                        2828+           SVC   C'B' .                          FREE STORAGE LIST
                                   2829+           DROP  2
00069C 4120 0184             00184 2830+XARETURN   LA    2,PSBSEM .                      WE ARE DONE, SO NOW SOMEONE
0006A0 0AE5                        2831+           SVC   C'V' .                           ELSE CAN COME IN
0006A2 1200                        2832+           LTR   0,0 .                           IS THIS FOR AUTOMATIC STORAGE?
0006A4 4770 10B0             006B0 2833+           BNZ   XABACK .                        IF NOT, RETURN NOW
0006A8 5060 F044             00044 2834+           ST    6,PCBASIZE .                    OTHERWISE, STORE SIZE AND
0006AC 50D0 F048             00048 2835+           ST    13,PCBAADDR .                    ADDRESS OF AUTOMATIC STORAGE
0006B0 0A6B                        2836+XABACK     SVC   C',' .                          LEAVE SMC SECTION
0006B2 8200 04D8       004D8       2837+           LPSW  RETURN .                        GET BACK JOJO
                                   2838+           DROP  4,7
```

```
            SAMPLE OPERATING SYSTEM       VERSION 2.00

   LOC    OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                      2840+**************************************************************************
                                      2841+*                                                                        *
                                      2842+*                              XF ROUTINE                                *
                                      2843+*                                                                        *
                                      2844+*      FUNCTION: TO FREE MEMORY                                          *
                                      2845+*     DATABASES: UPON ENTRY, REGISTER 2 CONTAINS ADDRESS XFX:            *
                                      2846+*                  XFX        DS 0D                                      *
                                      2847+*                  XFXSIZE    DS F     SIZE OF BLOCK TO BE FREED         *
                                      2848+*                  XFXADDR    DS A     ADDRESS OF FIRST BYTE OF BLOCK    *
                                      2849+* ROUTINES USED: XEXC, XP, XB, XV, XCOM                                  *
                                      2850+*     PROCEDURE: LOCK FSB SEMAPHORE; SEARCH FREE STORAGE LIST TO         *
                                      2851+*                FIND IF ANY FREE BLOCK CONTIGUOUSLY FOLLOWS OR          *
                                      2852+*                PRECEDES BLOCK TO BE FREED; IF THERE IS ANY,            *
                                      2853+*                COMPACT THEM INTO SINGLE BLOCK OF COMBINED SIZE;        *
                                      2854+*                USE XB TO CHAIN COMPACTED BLOCK ONTO FREE STORAGE       *
                                      2855+*                LIST; WAKEUP ALL PROCESSES WAITING ON MEMORY            *
                                      2856+*                SEMAPHORE; UNLOCK FSB SEMAPHORE; RETURN.                *
                                      2857+*  ERROR CHECKS: NONE                                                    *
                                      2858+*    INTERRUPTS: ON                                                      *
                                      2859+*   USER ACCESS: NO                                                      *
                                      2860+*                                                                        *
                                      2861+**************************************************************************
0006B6                                2863+XF        EQU   * .                THE XF ROUTINE, TO FREE STORAGE
0006B6                                2864+          USING *,1
0006B6 0A5A                           2865+          SVC   C'!' .             ENTER SMC SECTION
0006B8 1872                           2866+          LR    7,2
000000                                2867+          USING XFX,7 .            THE ARGUMENT LIST
0006BA 5830 7000         00000        2868+          L     3,XFXSIZE .        GET THE SIZE
0006BE 5840 7004         00004        2869+          L     4,XFXADDR .        AND THE ADDRESS
0006C2 1853                           2870+          LR    5,3 .              GET ADDRESS OF THE END OF THE
0006C4 1A54                           2871+          AR    5,4 .               BLOCK TO BE FREED
0006C6 4120 0184         00184        2872+          LA    2,FSBSEM .         LOCK FSBSEM
0006CA 0AD7                           2873+          SVC   C'P'
0006CC 4180 0180         00180        2874+          LA    8,FSBPTR .         START LOOKING DOWN THE FREE
0006D0 5860 0180         00180        2875+          L     6,FSBPTR .          STORAGE LIST, FOR COMPACTION
000000                                2876+          USING FSB,6
0006D4 1266                           2877+XFLOOP    LTR   6,6 .              ARE WE THROUGH?
0006D6 4780 105E         00714        2878+          BZ    XFLINK .           IF SO, JUST ADD IT ON
0006DA 5890 6000         00000        2879+          L     9,FSBNEXT .        IF NOT. GET THE NEXT PTR
0006DE 1965                           2880+          CR    6,5 .              IS THIS BLOCK RIGHT AFTER OURS?
0006E0 4770 103A         006F0        2881+          BNE   XFTHEN .           IF NOT, OK. BUT IF IT IS,
0006E4 5098 0000         00000        2882+          ST    9,0(8) .           WE CAN COMPACT. SO UNCHAIN IT
0006E8 5A30 6004         00004        2883+          A     3,FSBSIZE .        AND REMEMBER THE NEW SIZE
0006EC 47F0 1050         00706        2884+          B     XFBACKUP .         AND ON TO THE NEXT
0006F0 18A6                           2885+XFTHEN    LR    10,6 .             MAYBE IT'S RIGHT BEFORE OURS
0006F2 5AA0 6004         00004        2886+          A     10,FSBSIZE .       GET ENDING ADDRESS OF FREE BLOCK
0006F6 19A4                           2887+          CR    10,4 .             IS IT RIGHT BEFORE OURS?
0006F8 4770 1052         00708        2888+          BNE   XFINC .            OH FUDGE! NO!
0006FC 5098 0000         00000        2889+          ST    9,0(8) .           IF SO, UNLINK IT
000700 1846                           2890+          LR    4,6 .              GET NEW BEGINNING LOCATION
000702 5A30 6004         00004        2891+          A     3,FSBSIZE .        AND NEW SIZE OF FREE BLOCK
000706 1868                           2892+XFBACKUP  LR    6,8 .              BACK UP ONE FSB
000708 4180 6000         00000        2893+XFINC     LA    8,FSBNEXT .        ON TO THE NEXT FSB
00070C 5860 6000         00000        2894+          L     6,FSBNEXT
```

```
         SAMPLE OPERATING SYSTEM     VERSION 2.00

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT    SOURCE STATEMENT

000710 47F0 101E            006D4  2895+             B      XFLOOP .                       TRY, TRY AGAIN
000714 4120 E048            00048  2896+XFLINK       LA     2,SATEMP .                     START TO CALL XB
000000                             2897+             USING  XBX,2
000718 5030 2000            00000  2898+             ST     3,XBXSIZE .                    STORE SIZE
00071C 5040 2004            00004  2899+             ST     4,XBXADDR .                    AND ADDRESS
000720 0AC2                        2900+             SVC    C'B' .                         LINK IT ONTO THE FSB CHAIN
000000                             2901+             USING  SM,2
000722 4120 018C            0018C  2902+             LA     2,MEMORY .                     GET VALUE OF MEMORY SEMAPHORE
000726 41B0 0001            00001  2903+             LA     11,1(0,0) .                    SUBTRACT FROM ONE; IT'S A HANDLE
00072A 5BB0 2000            00000  2904+             S      11,SMVAL .                     ON THE # OF PEOPLE WAITING
                                   2905+             DROP   2
00072E 46B0 1088            0073E  2906+XFVLOOP      BCT    11,XFVDO .                     LOOP IF ANYONE ELSE IS WAITING
000732 4120 0184            00184  2907+             LA     2,FSBSEM .                     WE'RE THROUGH SO
000736 0AE5                        2908+             SVC    C'V' .                         UNLOCK FSBSEM
000738 0A6B                        2909+             SVC    C',' .                         LEAVE SMC
00073A 8200 04D8      004D8        2910+             LPSW   RETURN .                       RETURN
00073E 0AE5                        2911+XFVDO        SVC    C'V' .                         WAKE SOMEONE UP
000740 47F0 1078            0072E  2912+             B      XFVLOOP .                      TRY AGAIN FOR ANOTHER
                                   2913+             DROP   6,7
```

```
          SAMPLE OPERATING SYSTEM     VERSION 2.00

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                   2915+**************************************************************************
                                   2916+*                                                                        *
                                   2917+*                              XB ROUTINE                                *
                                   2918+*                                                                        *
                                   2919+*       FUNCTION: TO CHAIN A STORAGE BLOCK ONTO FREE STORAGE LIST        *
                                   2920+*      DATABASES: UPON ENTRY, REGISTER 2 CONTAINS ADDRESS XBX:           *
                                   2921+*                   XBX        DS 0D                                     *
                                   2922+*                   XBXSIZE    DS F     SIZE OF BLOCK                    *
                                   2923+*                   XBXADDR    DS A     ADDRESS OF FIRST BYTE OF BLOCK   *
                                   2924+*  ROUTINES USED: NONE                                                   *
                                   2925+*      PROCEDURE: SEARCH FREE STORAGE LIST TO FIND WHERE TO INSERT       *
                                   2926+*                 FREE BLOCK IN ORDER OF INCREASING SIZE; FORMAT         *
                                   2927+*                 BLOCK LIKE A FSB; INSERT; RETURN.                      *
                                   2928+*   ERROR CHECKS: NONE                                                   *
                                   2929+*     INTERRUPTS: OFF                                                    *
                                   2930+*    USER ACCESS: NO                                                     *
                                   2931+*       COMMENTS: SINCE  XB ROUTINE ONLY CALLED BY XA AND XF, FSB        *
                                   2932+*                 SEMAPHORE IS ALREADY LOCKED.                           *
                                   2933+*                                                                        *
                                   2934+**************************************************************************
000744                             2936+XB         EQU   *
000744                             2937+           USING *,1
000000                             2938+           USING XBX,2 .                      ARGUMENT LIST
000744 5830 2000          00000    2939+           L     3,XBXSIZE .                  GET THE SIZE
000748 5840 2004          00004    2940+           L     4,XBXADDR .                  AND THE ADDRESS
00074C 4180 0180          00180    2941+           LA    8,FSBPTR .                   START LOOKING DOWN THE CHAIN
000750 5860 0180          00180    2942+           L     6,FSBPTR
000754 1266                        2943+           LTR   6,6 .                        IF ZERO POINTER, WE ARE AT
000756 4780 102C          00770    2944+           BZ    XBINSERT .                   END OF CHAIN ALREADY
000000                             2945+           USING FSB,6
00075A 5930 6004          00004    2946+XBLOOP     C     3,FSBSIZE .                  IF THE SIZE OF OURS IS LESS,
00075E 47D0 102C          00770    2947+           BNP   XBINSERT .                    TIME TO INSERT
000762 4180 6000          00000    2948+           LA    8,FSBNEXT .                  ELSE GO ON TO THE NEXT
000766 5860 6000          00000    2949+           L     6,FSBNEXT
00076A 1266                        2950+           LTR   6,6 .                        IF NOT ALREADY THROUGH
00076C 4770 1016          0075A    2951+           BNZ   XBLOOP .                     BRANCH BACK
000770 5048 0000          00000    2952+XBINSERT   ST    4,0(8) .                     NOW, LINK OURS ON
                                   2953+           DROP  6
000000                             2954+           USING FSB,4
000774 5060 4000          00000    2955+           ST    6,FSBNEXT .                  MAKE OURS POINT TO THE NEXT
000778 5030 4004          00004    2956+           ST    3,FSBSIZE .                  WITH THE RIGHT SIZE
00077C 8200 04D8    004D8          2957+           LPSW  RETURN .                     AND RETURN
                                   2958+           DROP  2,4
```

```
        SAMPLE OPERATING SYSTEM     VERSION 2.00

 LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                  2960+***********************************************************************
                                  2961+*                                                                     *
                                  2962+*                           XC ROUTINE                                *
                                  2963+*                                                                     *
                                  2964+*        FUNCTION: TO CREATE A PROCESS                                *
                                  2965+*       DATABASES: UPON ENTRY, REGISTER 2 CONTAINS ADDRESS XCX:       *
                                  2966+*                    XCX        DS 0D                                 *
                                  2967+*                    XCXNAME    DS CL8  NAME OF PROCESS TO BE CREATED *
                                  2968+*   ROUTINES USED: XEXC, XCOM, XN, XA, XI, XQUE                       *
                                  2969+*       PROCEDURE: USE XA TO ALLOCATE NEW PCB; PLACE XCXNAME IN PCB;  *
                                  2970+*                  INITIALIZE SEMAPHORES; STOP; BLOCK; OUT OF SMC;    *
                                  2971+*                  CALL XI TO LINK PCB INTO PCB CHAINS; RETURN.       *
                                  2972+*    ERROR CHECKS: IF NAME ALREADY USED IN THIS GROUP, XQUE ENTERED.  *
                                  2973+*      INTERRUPTS: ON                                                 *
                                  2974+*     USER ACCESS: YES                                                *
                                  2975+*                                                                     *
                                  2976+***********************************************************************
000780                            2978+XC        EQU   * .                     THE XC ROUTINE: CREATE A PROCESS
000780                            2979+          USING *,1
000780 1872                       2980+          LR    7,2
000000                            2981+          USING XCX,7 .                 ARGUMENT LIST
000782 4120 E048            00048 2982+          LA    2,SATEMP .              READY TO MAKE CALLS OUT
000000                            2983+          USING XNX,2 .                 A XN-LIKE ARGUMENT LIST
000786 D207 2000 7000 00000 00000 2984+          MVC   XNXNAME,XCXNAME .       GET THE NAME
00078C 0AD5                       2985+          SVC   C'N' .                  AND CALL TO FIND THE PCB
00078E D503 2008 16B0 00008 00E30 2986+          CLC   XNXADDR,=A(0) .         SEE IF THERE
000794 4770 1044            007C4 2987+          BNE   XCERR .                 IF ALREADY EXISTS, BAD
000798 0A5A                       2988+          SVC   C'!' .                  ENTER SMC SECTION
                                  2989+          DROP  2
000000                            2990+          USING XAX,2 .                 READY TO CALL XA
00079A D203 2000 16B4 00000 00E34 2991+          MVC   XAXSIZE,=A(LENPCB) .    WE KNOW THE SIZE
0007A0 D203 2008 16B8 00008 00E38 2992+          MVC   XAXALGN,=F'8' .         AND THE ALIGNMENT
0007A6 0AC1                       2993+          SVC   C'A' .                  SO CALL
0007A8 5820 2004            00004 2994+          L     2,XAXADDR .             FIND THE ADDRESS
                                  2995+          DROP  2,15
000000                            2996+          USING PCB,2 .                 FILL IN THE PCB
0007AC D207 2000 7000 00000 00000 2997+          MVC   PCBNAME,XCXNAME .       GIVE IT A NAME
0007B2 92FF 2018            00018 2998+          MVI   PCBSTOPT,X'FF' .        IT'S STOPPED
0007B6 D232 2019 19F9 00019 01179 2999+          MVC   PCBBLOKT(PCBISA-PCBBLOKT),TEMPLATE+1 INITIALIZE PCB
0007BC 0AC9                       3000+          SVC   C'I' .                  THREAD IT ON
0007BE 0A6B                       3001+          SVC   C',' .                  LEAVE SMC SECTION
0007C0 8200 04D8            004D8 3002+          LPSW  RETURN .                AND RETURN
0007C4 0A6F                       3003+XCERR     SVC   C'?' .                  IF ALREADY EXISTS, ERROR
                                  3004+          DROP  2,7
```

```
                SAMPLE OPERATING SYSTEM         VERSION 2.00

LOC    OBJECT CODE      ADDR1 ADDR2  STMT   SOURCE STATEMENT
                                     3006+*******************************************************************
                                     3007+*                                                                  *
                                     3008+*                          XD ROUTINE                              *
                                     3009+*                                                                  *
                                     3010+*      FUNCTION: TO DESTROY A PROCESS                              *
                                     3011+*     DATABASES: UPON ENTRY, REGISTER 2 CONTAINS ADDRESS XDX:      *
                                     3012+*                  XDX         DS OD                               *
                                     3013+*                  XDXNAME     DS CL8  NAME OF PROCESS TO BE DESTROYED *
                                     3014+* ROUTINES USED: XEXC, XJ, XS, XN, XF, XCOM, XQUE                  *
                                     3015+*     PROCEDURE: USE XN TO FIND PCB FOR PROCESS TO BE DESTROYED;   *
                                     3016+*                USE XJ TO UNLOCK PCB FROM PROCESS CHAINS; IF ANY  *
                                     3017+*                MESSAGES FOR THIS PROCESS, FREE STORAGE FOR THEM; *
                                     3018+*                IF THERE IS ANY AUTOMATIC STORAGE, FREE IT;       *
                                     3019+*                FREE STORAGE FOR PCB; RETURN.                     *
                                     3020+*  ERROR CHECKS: IF NAME DOESN'T EXIST OR PROCESS NOT STOPPED,     *
                                     3021+*                XQUE ENTERED.                                     *
                                     3022+*    INTERRUPTS: ON                                                *
                                     3023+*   USER ACCESS: YES                                               *
                                     3024+*                                                                  *
                                     3025+*******************************************************************
0007C6                               3027+XD        EQU   * .                    XD ROUTINE: DESTROY A PROCESS
0007C6                               3028+          USING *,1
0007C6 1872                          3029+          LR    7,2
000000                               3030+          USING XDX,7 .                ARG LIST
0007C8 4120 E048              00048  3031+          LA    2,SATEMP .             READY TO CALL OUT
000000                               3032+          USING XNX,2 .                WILL CALL XN
0007CC D207 2000 7000 00000 00000    3033+          MVC   XNXNAME,XDXNAME .      GET NAME
0007D2 0AD5                          3034+          SVC   C'N' .                 AND CALL
0007D4 5820 2008              00008  3035+          L     2,XNXADDR .            GET ADDRESS
                                     3036+          DROP  2
0007D8 1222                          3037+          LTR   2,2 .                  IF ADDRESS IS NULL,
0007DA 4780 107A              00840  3038+          BZ    XDERR .                IT'S AN ERROR
000000                               3039+          USING PCB,2
0007DE 95FF 2018        00018        3040+          CLI   PCBSTOPT,X'FF' .       IF NOT STOPPED,
0007E2 4770 107A              00840  3041+          BNE   XDERR .                IT'S AN ERROR.
0007E6 0A5A                          3042+          SVC   C'!' .                 ENTER SMC SECTION
                                     3043+          DROP  2
000000                               3044+          USING PCB,15
0007E8 0AD1                          3045+          SVC   C'J' .                 ELSE UNTHREAD THE ENTRY
0007EA 1882                          3046+          LR    8,2 .                  REMEMBER THE PCB PTR
0007EC 4120 E048              00048  3047+          LA    2,SATEMP .             READY TO CALL OUT AGAIN
000000                               3048+          USING PCB,8
                                     3049+          DROP  15
0007F0 5890 802C              0002C  3050+          L     9,PCBFM .              GET FIRST MESSAGE
0007F4 1299                          3051+XDLOOP    LTR   9,9 .                  ANY MORE MESSAGES?
0007F6 4780 1054              0081A  3052+          BZ    XDCHECK .              IF NOT, FINISH UP
000000                               3053+          USING MSG,9
0007FA 58A0 9004              00004  3054+          L     10,MSGNEXT .           ELSE REMEMBER NEXT
0007FE 58B0 9008              00008  3055+          L     11,MSGSIZE .           GET THE SIZE
000802 41BB 000F              0000F  3056+          LA    11,15(11) .             AND MAKE IT SOME NUMBER
000806 54B0 1676              00E3C  3057+          N     11,=F'-8' .             OF DOUBLEWORDS
000000                               3058+          USING XFX,2
00080A 5090 2004              00004  3059+          ST    9,XFXADDR .            FREE THE LOCATION
00080E 50B0 2000              00000  3060+          ST    11,XFXSIZE .            THE NUMBER OF WORDS
```

```
        SAMPLE OPERATING SYSTEM      VERSION 2.00

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

000812 0AC6                        3061+          SVC   C'F' .                          DO IT
000814 189A                        3062+          LR    9,10 .                          ON TO THE NEXT
000816 47F0 102E            007F4  3063+          B     XDLOOP .                        GET NEXT MESSAGE
00081A D503 8048 166A 00048 00E30  3064+XDCHECK   CLC   PCBAADDR(4),=A(0) .             HAS AUTOMATIC STORAGE BEEN
000820 4780 1068            0082E  3065+          BE    XDTHEN .                         ALLOCATED? IF NOT, GO FINISH UP
000824 4120 8044            00044  3066+          LA    2,PCBASIZE .                    SET UP AGRUMENT LIST
000828 0AC6                        3067+          SVC   C'F' .                          FREE IT
00082A 4120 E048            00048  3068+          LA    2,SATEMP .                      RESET REGISTER 2
00082E 5080 2004            00004  3069+XDTHEN    ST    8,XFXADDR .                     READY TO FREE THE PCB
000832 D203 2000 166E 00000 00E34  3070+          MVC   XFXSIZE,=A(LENPCB) .            THE SIZE
000838 0AC6                        3071+          SVC   C'F' .                          FREE IT
00083A 0A6B                        3072+          SVC   C',' .                          LEAVE SMC
00083C 8200 04D8      004D8        3073+          LPSW  RETURN .                        AND RETURN
000840 0A6F                        3074+XDERR     SVC   C'?' .                          IF PROCESS DOES NOT EXIST
                                   3075+          DROP  2,7,9,8
000000                             3076+          USING PCB,15

                                   3078+*****************************************************************
                                   3079+*                                                               *
                                   3080+*                          XH ROUTINE                           *
                                   3081+*                                                               *
                                   3082+*       FUNCTION: TO HALT A JOB                                 *
                                   3083+*      DATABASES: NONE                                          *
                                   3084+*  ROUTINES USED: XS, XR                                        *
                                   3085+*      PROCEDURE: SEND MESSAGE TO SUPERVISOR PROCESS FOR THIS JOB *
                                   3086+*                 INDICATING NORMAL TERMINATION; TRIES TO READ  *
                                   3087+*                 MESSAGES FOREVER LOOPING; BLOCKS ITSELF, THEREBY *
                                   3088+*                 NEVER RETURNING.                              *
                                   3089+*   ERROR CHECKS: NONE                                          *
                                   3090+*     INTERRUPTS: ON                                            *
                                   3091+*    USER ACCESS: YES                                           *
                                   3092+*       COMMENTS: USER NORMALLY USES THIS ROUTINE TO END A JOB. *
                                   3093+*                                                               *
                                   3094+*****************************************************************
000842                             3096+XH        EQU   * .                             THE XH ROUTINE: HALT A JOB
000842                             3097+          USING *,1
000842 4120 1012            00854  3098+          LA    2,XHMSG1 .                      SEND A MESSAGE TO *IBSUP
000846 0AE2                        3099+          SVC   C'S' .                          SEND IT
000848 4120 102A            0086C  3100+XHLOOP    LA    2,XHMSG2 .                      READY TO HEAR A REPLY
00084C 0AD9                        3101+          SVC   C'R' .                           WHICH NEVER COMES
00084E 47F0 1006            00848  3102+          B     XHLOOP .                        BUT IF IT DOES WERE READY
000854                             3103+          DS    0F
000854 5CC9C2E2E4D74040            3104+XHMSG1    DC    CL8'*IBSUP' .                   SAY TO *IBSUP
00085C 0000000C                    3105+,         DC    F'12' .                         TWELVE CHARACTERS
000860 D7D9D6C7D9C1D440            3106+          DC    CL12'PROGRAM HALT' .            SAYING WERE OK
00086C                             3107+XHMSG2    DS    CL8 .                           WHO SENDS US A MESSAGE
000874 00000001                    3108+          DC    F'1' .                          ONE CHARACTER
000878                             3109+          DS    CL1,0H .                        WHICH GOES HERE
```

```
             SAMPLE OPERATING SYSTEM      VERSION 2.00

   LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT
                                    3111+***********************************************************************
                                    3112+*                                                                     *
                                    3113+*                            XI ROUTINE                               *
                                    3114+*                                                                     *
                                    3115+*      FUNCTION: TO CHAIN A PCB ONTO PROCESS CHAINS                   *
                                    3116+*     DATABASES: UPON ENTRY, REGISTER 2 CONTAINS ADDRESS OF A PCB     *
                                    3117+* ROUTINES USED: NONE                                                 *
                                    3118+*     PROCEDURE: POINTER USED TO CHAIN PCB INTO ALL PCB CHAIN AND     *
                                    3119+*                THIS GROUP CHAIN RIGHT AFTER RUNNING PCB; RETURN.    *
                                    3120+*  ERROR CHECKS: NONE                                                 *
                                    3121+*    INTERRUPTS: OFF                                                  *
                                    3122+*   USER ACCESS: NO                                                   *
                                    3123+*                                                                     *
                                    3124+***********************************************************************
 00087A                             3126+XI        EQU   * .                       THE XI ROUTINE: THREAD IN A PCB
 00087A                             3127+          USING *,1
 00087A 58A0 F010            00010  3128+          L     10,PCBNPALL .             GET THE NEXT 'ALL' PCB
 00087E 5020 F010            00010  3129+          ST    2,PCBNPALL .              STORE THIS PCB RIGHT AFTER MINE
                                    3130+          DROP  15
 000000                             3131+          USING PCB,10
 000882 5020 A014            00014  3132+          ST    2,PCBLPALL .              THE NEXT ONE DOWN POINTS BACK
                                    3133+          DROP  10
 000000                             3134+          USING PCB,2
 000886 50F0 2014            00014  3135+          ST    15,PCBLPALL .             THIS PCB POINTS BACK
 00088A 50A0 2010            00010  3136+          ST    10,PCBNPALL .              AND FORWARD
                                    3137+          DROP  2
 000000                             3138+          USING PCB,15
 00088E 58A0 F008            00008  3139+          L     10,PCBNPTG .              GET NEXT "THIS GROUP" PCB
 000892 5020 F008            00008  3140+          ST    2,PCBNPTG .               RUNNING PCB POINTS TO NEW MEMBER
                                    3141+          DROP  15 .                       OF PROCESS GROUP
 000000                             3142+          USING PCB,10
 000896 5020 A00C            0000C  3143+          ST    2,PCBLPTG .               NEXT PCB DOWN POINTS BACK
                                    3144+          DROP  10
 000000                             3145+          USING PCB,2
 00089A 50F0 200C            0000C  3146+          ST    15,PCBLPTG .              AND WE POINT BACKWARD
 00089E 50A0 2008            00008  3147+          ST    10,PCBNPTG .              AND FORWARD
                                    3148+          DROP  2
 0008A2 8200 04D8      004D8        3149+          LPSW  RETURN .                  RETURN
 000000                             3150+          USING PCB,15
```

```
             SAMPLE OPERATING SYSTEM       VERSION 2.00

  LOC   OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                    3152+***********************************************************************
                                    3153+*                                                                     *
                                    3154+*                             XJ ROUTINE                              *
                                    3155+*                                                                     *
                                    3156+*       FUNCTION: TO UNCHAIN A PCB FROM PROCESS CHAINS                *
                                    3157+*      DATABASES: UPON ENTRY, REGISTER 2 CONTAINS ADDRESS OF A PCB    *
                                    3158+*  ROUTINES USED: NONE                                               *
                                    3159+*      PROCEDURE: POINTERS TO PCB IN ALL PCB CHAIN AND THIS GROUP     *
                                    3160+*                 CHAIN MODIFIED WITHOUT FREEING STORAGE; RETURN.     *
                                    3161+*  ERROR CHECKS: NONE                                                 *
                                    3162+*    INTERRUPTS: OFF                                                  *
                                    3163+*   USER ACCESS: NO                                                  *
                                    3164+*                                                                     *
                                    3165+***********************************************************************
0008A6                              3167+XJ       EQU   * .                           THE XJ ROUTINE: UNTHREAD A PCB
0008A6                              3168+         USING *,1
                                    3169+         DROP  15
000000                              3170+         USING PCB,2
0008A6 58B0 2014          00014     3171+         L     11,PCBLPALL .               GET PRECEDING PCB
0008AA 58A0 2010          00010     3172+         L     10,PCBNPALL .                AND FOLLOWING ONE IN "ALL"
                                    3173+         DROP  2 .                            CHAIN
000000                              3174+         USING PCB,11
0008AE 50A0 B010          00010     3175+         ST    10,PCBNPALL .               LAST POINTS TO NEXT
                                    3176+         DROP  11
000000                              3177+         USING PCB,10
0008B2 50B0 A014          00014     3178+         ST    11,PCBLPALL .               NEXT POINTS TO LAST
                                    3179+         DROP  10
000000                              3180+         USING PCB,2
0008B6 58B0 200C          0000C     3181+         L     11,PCBLPTG .                REDO FOR THIS GROUP PCB CHAIN
0008BA 58A0 2008          00008     3182+         L     10,PCBNPTG
                                    3183+         DROP  2
000000                              3184+         USING PCB,11
0008BE 50A0 B008          00008     3185+         ST    10,PCBNPTG .                LAST POINTS TO NEXT
                                    3186+         DROP  11
000000                              3187+         USING PCB,10
0008C2 50B0 A00C          0000C     3188+         ST    11,PCBLPTG .                NEXT POINTS TO LAST
                                    3189+         DROP  10
0008C6 8200 04D8    004D8           3190+         LPSW  RETURN .                    AND RETURN
000000                              3191+         USING PCB,15
```

```
            SAMPLE OPERATING SYSTEM     VERSION 2.00

   LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT
                                     3193+**********************************************************************
                                     3194+*                                                                    *
                                     3195+*                            XN ROUTINE                              *
                                     3196+*                                                                    *
                                     3197+*       FUNCTION: TO FIND THE PCB FOR A PROCESS GIVEN ITS NAME ONLY  *
                                     3198+*      DATABASES: UPON ENTRY, REGISTER 2 CONTAINS ADDRESS XNX:       *
                                     3199+*                      XNX       DS OD                               *
                                     3200+*                      XNXNAME   DS CL8  NAME OF PROCESS             *
                                     3201+*                      XNXADDR   DS A    ADDRESS OF PCB              *
                                     3202+*  ROUTINES USED: NONE                                               *
                                     3203+*      PROCEDURE: SEARCH THIS GROUP PCB CHAIN FOR NAME; IF FOUND,    *
                                     3204+*                 STORE POINTER IN XNXADDR. IF NOT FOUND, STORE      *
                                     3205+*                 ZERO IN XNXADDR. RETURN.                           *
                                     3206+*   ERROR CHECKS: NONE                                               *
                                     3207+*     INTERRUPTS: OFF                                                *
                                     3208+*    USER ACCESS: YES                                                *
                                     3209+*                                                                    *
                                     3210+**********************************************************************
0008CA                               3212+XN         EQU   * .                         THE XN ROUTINE: FIND A NAMED PCB
0008CA                               3213+           USING *,1
000000                               3214+           USING XNX,2 .                     THE ARG LIST
0008CA 18AF                          3215+           LR    10,15 .                     FIRST PCB TO LOOK AT IS OURS
                                     3216+           DROP  15
000000                               3217+           USING PCB,10
0008CC 58A0 A008              00008  3218+XNLOOP     L     10,PCBNPTG .                LOOK AT NEXT PCB
0008D0 D507 A000 2000 00000   00000  3219+           CLC   PCBNAME,XNXNAME .           HAS IT THE RIGHT NAME?
0008D6 4780 101A              008E4  3220+           BE    XNFOUND .                   IF YES, OH JOY.
0008DA 19AF                          3221+           CR    10,15 .                     IF NOT, ARE WE THROUGH?
0008DC 4770 1002              008CC  3222+           BNE   XNLOOP .                    IF NOT, TRY NEXT PCB
0008E0 41A0 0000              00000  3223+           LA    10,0 .                      ELSE, IT'S NOT HERE
0008E4 50A0 2008              00008  3224+XNFOUND    ST    10,XNXADDR .                FOUND IT. SAY WHERE.
0008E8 8200 04D8        004D8        3225+           LPSW  RETURN .                    AND RETURN
                                     3226+           DROP  2,10
000000                               3227+           USING PCB,15
```

```
                SAMPLE OPERATING SYSTEM      VERSION 2.00
   LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                    3229+****************************************************************
                                    3230+*                                                              *
                                    3231+*                           XR ROUTINE                         *
                                    3232+*                                                              *
                                    3233+*       FUNCTION: TO READ A MESSAGE                            *
                                    3234+*      DATABASES: UPON ENTRY, REGISTER 2 CONTAINS ADDRESS XRX: *
                                    3235+*                   XRX       DS 0D                            *
                                    3236+*                   XRXNAME   DS CL8  NAME OF SENDER PROCESS   *
                                    3237+*                   XRXSIZE   DS F    SIZE OF MESSAGE TEXT     *
                                    3238+*                   XRXTEXT   DS C    TEXT OF MESSAGE          *
                                    3239+*  ROUTINES USED: XP, XEXC, XN, XCOM, XF                       *
                                    3240+*      PROCEDURE: USE XP ON MESSAGE SEMAPHORE RECEIVER TO SEE IF ANY *
                                    3241+*                 MESSAGES WAITING; IF NONE, PROCESS BLOCKED UNTIL *
                                    3242+*                 THERE IS ONE; LOCK MESSAGE CHAIN; REMOVE A MESSAGE *
                                    3243+*                 FROM CHAIN AND UNLOCK IT; MOVE TEXT OF MESSAGE, *
                                    3244+*                 PADDING WITH BLANKS OR TRUNCATING AS NECESSARY; *
                                    3245+*                 INDICATE CORRECT MESSAGE LENGTH AND NAME OF  *
                                    3246+*                 MESSAGE SENDER; FREE STORAGE USED TO HOLD MESSAGE, *
                                    3247+*                 AND RETURN.                                  *
                                    3248+*   ERROR CHECKS: NONE                                         *
                                    3249+*     INTERRUPTS: ON                                           *
                                    3250+*    USER ACCESS: YES                                          *
                                    3251+*                                                              *
                                    3252+****************************************************************
0008EC                              3254+XR         EQU   * .                     THE XR ROUTINE: READ A MESSAGE
0008EC                              3255+           USING *,1
0008EC 1872                         3256+           LR    7,2
000000                              3257+           USING XRX,7 .                 ARG LIST
0008EE 4120 F024              00024 3258+           LA    2,PCBMSR .              SEE IF MESSAGES WAITING
0008F2 0AD7                         3259+           SVC   C'P'
0008F4 0A5A                         3260+           SVC   C'!' .                  ENTER SMC SECTION
0008F6 4120 F01C              0001C 3261+           LA    2,PCBMSC .              THEN LOCK THE MESSAGE CHAIN
0008FA 0AD7                         3262+           SVC   C'P'
0008FC 5850 F02C              0002C 3263+           L     5,PCBFM .               GET THE FIRST MESSAGE
000000                              3264+           USING MSG,5
000900 D203 F02C 5004 0002C 00004   3265+           MVC   PCBFM,MSGNEXT .         REMEMBER THE NEXT
000906 0AE5                         3266+           SVC   C'V' .                  UNLOCK THE MESSAGE CHAIN
000908 5860 7008              00008 3267+           L     6,XRXSIZE .             GET THE BUFFER CAPACITY
00090C 5B60 1554              00E40 3268+           S     6,=F'2' .               MINUS 1, MINUS 1
000910 9240 700C        0000C       3269+           MVI   XRXTEXT,C' ' .          MOVE IN A BLANK
000914 4740 1030              0091C 3270+           BM    XRNOB
000918 4460 1080              0096C 3271+           EX    6,XRFILL .              THEN FILL THE REST WITH BLANKS
00091C 4166 0001              00001 3272+XRNOB      LA    6,1(6) .                THEN GET PROPER BUFFER COUNT
000920 5960 5008              00008 3273+           C     6,MSGSIZE .             COMPARE WITH MESSAGE LENGTH
000924 4740 1042              0092E 3274+           BL    XRTHEN .                IF LESS, HANDLE ACCORDINGLY
000928 5860 5008              00008 3275+           L     6,MSGSIZE .             ELSE COUNT FOR MVC IS MESSAGE
00092C 0660                         3276+           BCTR  6,0 .                    SIZE MINUS ONE
00092E 1266                         3277+XRTHEN     LTR   6,6 .                   ANY CHARACTERS TO MOVE?
000930 4740 104C              00938 3278+           BM    XRAPT .                 IF NOT, DON'T
000934 4460 1086              00972 3279+           EX    6,XRMOVE .              ELSE MOVE THEM
000938 4166 0001              00001 3280+XRAPT      LA    6,1(6) .                THEN GET LENGTH
00093C 5060 7008              00008 3281+           ST    6,XRXSIZE .             STORE IT
000940 58A0 5000              00000 3282+           L     10,MSGSENDR .           GET SENDER'S PCB
                                    3283+           DROP  15
```

```
              SAMPLE OPERATING SYSTEM       VERSION 2.00
  LOC   OBJECT CODE      ADDR1 ADDR2  STMT    SOURCE STATEMENT

000000                                3284+            USING PCB,10
000944 D207 7000 A000 00000 00000     3285+            MVC   XRXNAME,PCBNAME .        AND STORE SENDER'S NAME
00094A 5860 5008              00008   3286+            L     6,MSGSIZE .              GET SIZE OF MESSAGE TEXT,
00094E 4166 000C              0000C   3287+            LA    6,LENMSG(6) .             ADD SIZE OF MESSAGE BLOCK
000952 4166 0007              00007   3288+            LA    6,7(6) .                  AND TRUNCATE
000956 5460 1550              00E3C   3289+            N     6,=F'-8' .                UP
00095A 1825                           3290+            LR    2,5 .                     AND SET UP POINTER TO XFX
000000                                3291+            USING XFX,2
00095C 5050 2004              00004   3292+            ST    5,XFXADDR .              STORE ADDRESS
000960 5060 2000              00000   3293+            ST    6,XFXSIZE .              STORE SIZE
000964 0AC6                           3294+            SVC   C'F' .                   AND FREE THE MESSAGE BLOCK
000966 0A6B                           3295+            SVC   C',' .                   LEAVE SMC
000968 8200 04D8        004D8         3296+            LPSW  RETURN .                 AND RETURN
00096C D200 700D 700C 0000D 0000C     3297+XRFILL      MVC   XRXTEXT+1,XRXTEXT .      FILL WITH BLANKS
000972 D200 700C 500C 0000C 0000C     3298+XRMOVE      MVC   XRXTEXT,MSGTEXT .        MOVE TEXT
                                      3299+            DROP  2,5,7,10
000000                                3300+            USING PCB,15

                                      3302+*************************************************************************
                                      3303+*                                                                       *
                                      3304+*                                XS ROUTINE                             *
                                      3305+*                                                                       *
                                      3306+*       FUNCTION: TO SEND A MESSAGE                                     *
                                      3307+*      DATABASES: UPON ENTRY, REGISTER 2 CONTAINS ADDRESS XSX:          *
                                      3308+*                    XSX         DS QD                                  *
                                      3309+*                    XSXNAME     DS CL8   NAME OF TARGET PROCESS        *
                                      3310+*                    XSXSIZE     DS F     SIZE OF TEXT                  *
                                      3311+*                    XSXTEXT     DS C     TEXT OF MESSAGE               *
                                      3312+*  ROUTINES USED: XP, XV, XEXC, XCOM, XA, XN, XQUE                      *
                                      3313+*      PROCEDURE: USE XN TO GET POINTER TO PCB OF TARGET PROCESS;       *
                                      3314+*                 USE LENGTH OF MESSAGE AND XA TO ALLOCATE BLOCK FOR    *
                                      3315+*                 MESSAGE; LOCK MESSAGE CHAIN OF TARGET PROCESS;        *
                                      3316+*                 PUT MESSAGF BLOCK AT END OF CHAIN; STORE SENDER       *
                                      3317+*                 NAME, SIZE, AND TEXT OF MESSAGE; UNLOCK CHAIN;        *
                                      3318+*                 INDICATE MESSAGE CHAIN IS ONE LONGER; RETURN.         *
                                      3319+*   ERROR CHECKS: IF NO PROCESS BY GIVEN NAME, ENTER XQUE.              *
                                      3320+*     INTERRUPTS: ON                                                    *
                                      3321+*    USER ACCESS: YES                                                   *
                                      3322+*                                                                       *
                                      3323+*************************************************************************

000978                                3325+XS          EQU   * .                      THE XS ROUTINE: SEND MESSAGES
000978                                3326+            USING *,1
000978 1872                           3327+            LR    7,2
000000                                3328+            USING XSX,7 .                  ARG LIST
00097A 4120 E048              00048   3329+            LA    2,SATEMP .               READY TO CALL OUT
000000                                3330+            USING XNX,2 .                  ABOUT TO CALL XN
00097E D207 2000 7000 00000 00000     3331+            MVC   XNXNAME,XSXNAME .        GIVE NAME OF TARGET PROCESS
000984 0AD5                           3332+            SVC   C'N' .                   SEE WHERE IT IS
000986 5840 2008              00008   3333+            L     4,XNXADDR .              GET THE POINTER
00098A 1244                           3334+            LTR   4,4 .                    IS THERE INDEED ONE?
00098C 4780 108A              00A02   3335+            BZ    XSERR .                  IF NOT ,ERROR
000000                                3336+            USING PCB,4
```

```
            SAMPLE OPERATING SYSTEM     VERSION 2.00

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT    SOURCE STATEMENT

                                   3337+           DROP  2,15
000000                             3338+           USING XAX,2 .                  READY TO CALL XA
000990 0A5A                        3339+           SVC   C'!' .                   ENTERING SMC SECTION
000992 5830 7008            00008  3340+           L     3,XSXSIZE .              GET THE STATED SIZE
000996 4133 000C            0000C  3341+           LA    3,LENMSG(3) .            PLUS THE AMOUNT OF OVERHEAD
00099A 4133 0007            00007  3342+           LA    3,7(3) .                 AND TRUNCATE
00099E 5430 14C4            00E3C  3343+           N     3,=F'-8' .                UP
0009A2 5030 2000            00000  3344+           ST    3,XAXSIZE .              THAT'S THE SIZE OF THE REGION TO
0009A6 D203 2008 14C0 00008 00E38  3345+           MVC   XAXALGN,=F'8' .          ALLOCATE, ON A DOUBLEWORD BOUND
0009AC 0AC1                        3346+           SVC   C'A' .                   SO ALLOCATE ALREADY
0009AE 5850 2004            00004  3347+           L     5,XAXADDR .              GET THE ADDRESS
                                   3348+           DROP  2
0009B2 4120 401C            0001C  3349+           LA    2,PCBMSC .               GET THE MESSAGE CHAIN SEMAPHORE
0009B6 0AD7                        3350+           SVC   C'P' .                    AND LOCK IT
0009B8 4180 402C            0002C  3351+           LA    8,PCBFM .                THEN START DOWN THE MESSAGE
0009BC 5890 402C            0002C  3352+           L     9,PCBFM .                 CHAIN
000000                             3353+           USING MSG,9
0009C0 1299                        3354+XSLOOP     LTR   9,9 .                    ARE WE THROUGH?
0009C2 4780 105A            009D2  3355+           BZ    XSADD .                  IF SO ADD IT ON.
0009C6 4180 9004            00004  3356+           LA    8,MSGNEXT .              IF NOT, ON TO THE NEXT
0009CA 5890 9004            00004  3357+           L     9,MSGNEXT
0009CE 47F0 1048            009C0  3358+           B     XSLOOP .                 AND TRY AGAIN
0009D2 5058 0000            00000  3359+XSADD      ST    5,0(8) .                 CHAIN OURS ON THE END
                                   3360+           DROP  9
000000                             3361+           USING MSG,5
0009D6 D203 5004 14B8 00004 00E30  3362+           MVC   MSGNEXT,=A(0) .          SET NEXT POINTER NULL
0009DC 50F0 5000            00000  3363+           ST    15,MSGSENDR .            STORE THE SENDER
0009E0 5860 7008            00008  3364+           L     6,XSXSIZE .              GET THE TEXT LENGTH
0009E4 5060 5008            00008  3365+           ST    6,MSGSIZE .              AND STORE IT
0009E8 0660                        3366+           BCTR  6,0 .                    ONE LESS
0009EA 1266                        3367+           LTR   6,6 .                    TEST LENGTH
0009EC 4740 107C            009F4  3368+           BM    XSAFT .                  IF ZERO, NOTHING TO MOVE
0009F0 4460 108C            00A04  3369+           EX    6,XSMOVE .               ELSE, MOVE IT
0009F4 0AE5                        3370+XSAFT      SVC   C'V' .                   UNLOCK THE MESSAGE CHAIN
0009F6 4120 4024            00024  3371+           LA    2,PCBMSR .               THEN SAY THERE'S
0009FA 0AE5                        3372+           SVC   C'V' .                    ONE MORE MESSAGE
0009FC 0A6B                        3373+           SVC   C',' .                   LEAVE SMC SECTION
0009FE 8200 04D8      004D8        3374+           LPSW  RETURN .                 AND RETURN
000A02 0A6F                        3375+XSERR      SVC   C'?'
000A04 D200 500C 700C 0000C 0000C  3376+XSMOVE     MVC   MSGTEXT,XSXTEXT .        THE MOVE FOR THE TEXT
                                   3377+           DROP  4,5,7
000000                             3378+           USING PCB,15
```

```
        SAMPLE OPERATING SYSTEM     VERSION 2.00

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                   3380+***********************************************************************
                                   3381+*                                                                     *
                                   3382+*                              XY ROUTINE                             *
                                   3383+*                                                                     *
                                   3384+*       FUNCTION: TO START A PROCESS                                  *
                                   3385+*      DATABASES: UPON ENTRY, REGISTER 2 CONTAINS ADDRESS XYX:        *
                                   3386+*                   XYX        DS OD                                  *
                                   3387+*                   XYXNAME   DS CL8  NAME OF PROCESS TO BE STARTED  *
                                   3388+*                   XYXADDR   DS A     STARTING ADDRESS OF PROCESS    *
                                   3389+*  ROUTINES USED: XN, XEXC, XCOM, XQUE                                *
                                   3390+*      PROCEDURE: USE XN TO GET POINTER TO PCB OF PROCESS TO BE       *
                                   3391+*                 STARTED; STORE IN PCB INTERRUPT SAVE AREA REGISTERS *
                                   3392+*                 AND PSW WITH STARTING ADDRESS AS SENT FROM STARTING *
                                   3393+*                 PROCESS; STOPPED BIT TURNED OFF; RETURN.            *
                                   3394+*   ERROR CHECKS: IF NO PROCESS BY GIVEN NAME, XQUE ENTERED.          *
                                   3395+*     INTERRUPTS: OFF                                                 *
                                   3396+*    USER ACCESS: YES                                                 *
                                   3397+*                                                                     *
                                   3398+***********************************************************************
000A0A                             3400+XY          EQU   * .                          THE XY ROUTINE: START A PROCESS
000A0A                             3401+            USING *,1
000A0A 1872                        3402+            LR    7,2
000000                             3403+            USING XYX,7 .                      THE ARG LIST
000A0C 4120 E048             00048 3404+            LA    2,SATEMP .                   READY TO CALL OUT
000000                             3405+            USING XNX,2
000A10 D207 2000 7000 00000 00000  3406+            MVC   XNXNAME,XYXNAME .            GIVE XN A NAME
000A16 0AD5                        3407+            SVC   C'N' .                       CALL XN
000A18 58A0 2008             00008 3408+            L     10,XNXADDR .                 WHERE IS THE PCB?
000A1C 12AA                        3409+            LTR   10,10 .                       OR IS THERE ONE?
000A1E 4780 1036             00A40 3410+            BZ    XYERR .                      IF NOT, OH HISS BOO
                                   3411+            DROP  2,14,15
000000                             3412+            USING PCB,10
000A22 41D0 A04C             0004C 3413+            LA    13,PCBISA .                  GET INTO THAT PCB'S ISA
000000                             3414+            USING SA,13
000A26 D207 D000 E000 00000 00000  3415+            MVC   SAPSW,(SAPSW-SA)(14) .       GIVE IT THE CALLER'S PSW
000A2C D202 D005 7009 00005 00009  3416+            MVC   SAPSW+5(3),XYXADDR+1 .       BUT AT THE REQUESTED ADDRESS
000A32 D23F D008 E008 00008 00008  3417+            MVC   SAREGS,(SAREGS-SA)(14)  .GIVE IT HIS REGISTERS
000A38 9200 A018       00018       3418+            MVI   PCBSTOPT,X'00' .             IT'S NO LONGER STOPPED
000A3C 8200 04D8       004D8       3419+            LPSW  RETURN .                     AND RETURN
000A40 0A6F                        3420+XYERR       SVC   C'?' .                       WE DONE BAD
                                   3421+            DROP  7,10,13
000000                             3422+            USING SA,14
000000                             3423+            USING PCB,15
```

```
          SAMPLE OPERATING SYSTEM     VERSION 2.00

     LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                      3425+***********************************************************************
                                      3426+*                                                                     *
                                      3427+*                            XZ ROUTINE                               *
                                      3428+*                                                                     *
                                      3429+*      FUNCTION: TO STOP A PROCESS                                    *
                                      3430+*     DATABASES: UPON ENTRY, REGISTER 2 CONTAINS ADDRESS XZX:         *
                                      3431+*                  XZX        DS OD                                   *
                                      3432+*                  XZXNAME    DS CL8  NAME OF PROCESS TO BE STOPPED   *
                                      3433+* ROUTINES USED: XN, XEXC, XCOM, XQUE, XP                             *
                                      3434+*     PROCEDURE: CHECK THAT USER PROCESS CAN'T STOP SYSTEM            *
                                      3435+*                PROCESS; USE XN TO GET PCB POINTER; IF IN SMC, SET   *
                                      3436+*                STOP WAITING BIT AND BLOCK SELF UNTIL STOP           *
                                      3437+*                PERFORMED; ELSE SET STOPPED BIT, AND RETURN.         *
                                      3438+*  ERROR CHECKS: IF NO PROCESS BY GIVEN NAME OR USER TRIES TO         *
                                      3439+*                STOP A SYSTEM PROCESS, XQUE ENTERED.                 *
                                      3440+*    INTERRUPTS: ON·                                                  *
                                      3441+*   USER ACCESS: YES                                                  *
                                      3442+*                                                                     *
                                      3443+***********************************************************************
   000A42                             3445+XZ        EQU   * .                       THE XZ ROUTINE: STOP A PROCESS
   000A42                             3446+          USING *,1
   000A42 1872                        3447+          LR    7,2
   000000                             3448+          USING XZX,7 .                   ARG LIST
   000A44 955C F000      00000        3449+          CLI   PCBNAME,C'*' .            IS STOPPER A * PROCESS?
   000A48 4780 1012            00A54  3450+          BE    XZFINE .                  THAT'S OK
   000A4C 955C 7000      00000        3451+          CLI   XZXNAME,C'*' .            IF NOT,IS STOPPEE A * ?
   000A50 4780 104A            00A8C  3452+          BE    XZERR .                   CAN'T DO THAT
   000A54 4120 E048            00048  3453+XZFINE    LA    2,SATEMP .                READY TO CALL OUT
   000000                             3454+          USING XNX,2 .                   WILL CALL XN
   000A58 D207 2000 7000 00000 00000  3455+          MVC   XNXNAME,XZXNAME .         GIVE IT THE NAME
   000A5E 0AD5                        3456+          SVC   C'N' .                    AND DO THE CALL
   000A60 58A0 2008            00008  3457+          L     10,XNXADDR .              GET THE PCB'S ADDRESS
   000A64 12AA                        3458+          LTR   10,10 .                   SEE IF NULL
   000A66 4780 104A            00A8C  3459+          BZ    XZERR .                   IF SO, ERROR.
   000A6A 0A5A                        3460+          SVC   C'!' .                    ENTER SMC
                                      3461+          DROP  2,15
   000000                             3462+          USING PCB,10
   000A6C 9500 A01A      0001A        3463+XZSTOP    CLI   PCBINSMC,X'00' .          SEE IF IN SMC
   000A70 4770 103C            00A7E  3464+          BNE   XZINSMC .                 IF SO, BAD
   000A74 92FF A018      00018        3465+          MVI   PCBSTOPT,X'FF' .          ELSE JUST STOP IT
   000A78 0A6B                        3466+          SVC   C',' .                    LEAVE SMC
   000A7A 8200 04D8      004D8        3467+          LPSW  RETURN .                  AND RETURN
   000A7E 92FF A01B      0001B        3468+XZINSMC   MVI   PCBSW,X'FF' .             IF IN SMC, SAY STOP WAITING
   000A82 4120 A034            00034  3469+          LA    2,PCBSRS .                AND STOP OURSELVES AGAINST
   000A86 0AD7                        3470+          SVC   C'P' .                     A SEMAPHORE
   000A88 47F0 102A            00A6C  3471+          B     XZSTOP .                  THEN WE CAN REALLY STOP IT
   000A8C 0A6F                        3472+XZERR     SVC   C'?' .                    AN ERROR
                                      3473+          DROP  10,7
   000000                             3474+          USING PCB,15
```

```
      SAMPLE OPERATING SYSTEM.     VERSION 2.00

 LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                  3476+*********************************************************************
                                  3477+*                                                                   *
                                  3478+*                              XQUE ROUTINE                         *
                                  3479+*                                                                   *
                                  3480+*       FUNCTION: TO SIGNAL ERROR CONDITION                         *
                                  3481+*      DATABASES: NONE                                              *
                                  3482+*  ROUTINES USED: XR, XS                                            *
                                  3483+*      PROCEDURE: SEND MESSAGE TO SUPERVISOR PROCESS FOR THIS JOB   *
                                  3484+*                 INDICATING ABNORMAL TERMINATION; TRY TO READ      *
                                  3485+*                 MESSAGES, FOREVER LOOPING; BLOCK ITSELF, THEREBY  *
                                  3486+*                 NEVER RETURNING.                                  *
                                  3487+*   ERROR CHECKS: NONE                                              *
                                  3488+*     INTERRUPTS: OFF                                               *
                                  3489+*    USER ACCESS: YES                                               *
                                  3490+*                                                                   *
                                  3491+*********************************************************************
000A8E                            3493+XQUE     EQU   * .                    THE XQUE ROUTINE: ERROR!
000A8E                            3494+         USING *,1
000A8E 4120 1012            00AA0 3495+         LA    2,XQUEM1 .             SEND AN ERROR MESSAGE TO *IBSUP
000A92 0AE2                       3496+         SVC   C'S'
000A94 4120 102A            00AB8 3497+XQUELOOP LA    2,XQUEM2 .             WAIT FOR REPLY
000A98 0AD9                       3498+         SVC   C'R'
000A9A 47F0 1006            00A94 3499+         B     XQUELOOP .             BUT IGNORE IT
000AA0                            3500+         DS    0F
000AA0 5CC9C2E2E4D74040           3501+XQUEM1   DC    CL8'*IBSUP'
000AA8 0000000C                   3502+         DC    F'12'
000AAC D7D9D6C7D9C1D440           3503+         DC    CL12'PROGRAM FLOP'
000AB8                            3504+XQUEM2   DS    CL8
000AC0 00000001                   3505+         DC    F'1'
000AC4                            3506+         DS    CL1,0H
                                  3507+         DROP  14,15
```

```
               SAMPLE OPERATING SYSTEM       VERSION 2.00
    LOC   OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT
                                      3509+**********************************************************************
                                      3510+*                                                                    *
                                      3511+*                        INPUT/OUTPUT ROUTINES                       *
                                      3512+*                                                                    *
                                      3513+**********************************************************************

                                      3515+**********************************************************************
                                      3516+*                                                                    *
                                      3517+*               SYSTEM SUPPLIED DEVICE HANDLER FOR READERS           *
                                      3518+*                                                                    *
                                      3519+**********************************************************************

  000AC6                              3521+RDRHANDL EQU   * .                          THE READER HANDLER
  000000                              3522+         USING UCB,3 .                      STARTED WITH REG3 -> UCB
  000AC6 0510                         3523+         BALR  1,0
  000AC8                              3524+         USING *,1 .                        ESTABLISH ADDRESSING
  000AC8 4120 1140           00C08    3525+         LA    2,RDRHSEM .                  LOCK OURSELVES UNTIL WE SET UP
  000ACC 0AD7                         3526+         SVC   C'P' .                        AN AUTOMATIC STORAGE AREA
  000ACE 4120 1150           00C18    3527+         LA    2,RDRHAAS .                  READY TO ALLOCATE
  000000                              3528+         USING XAX,2
  000AD2 0AC5                         3529+         SVC   C'E' .                       ALLOCATE
  000AD4 58C0 2004           00004    3530+         L     12,XAXADDR .                 GET A PTR
                                      3531+         DROP  2
  000AD8 4120 1140           00C08    3532+         LA    2,RDRHSEM .                  AND UNLOCK OURSELVES
  000ADC 0AE5                         3533+         SVC   C'V'
  000ADE 8840 0010           00010    3534+         SRL   4,16 .                       SHIFT KEY
  000AE2 1BAA                         3535+         SR    10,10 .                      CLEAR REG 10
  000000                              3536+         USING RDRHAS,12 .                  AUTOMATIC AREA
  000AE4 9200 C07A     0007A          3537+         MVI   JOBBIT,X'00' .               INITIALIZE
  000AE8 4160 C000           00000    3538+         LA    6,RDRHCCB .                  GET PTR TO CCB
  000AEC 4120 C008           00008    3539+RDRHLOOP LA    2,RDRHMSG .                  TRY TO READ A MESSAGE
  000000                              3540+         USING XRX,2
  000AF0 D203 2008 1370 00008 00E38   3541+         MVC   XRXSIZE,=F'8' .              WE CAN TAKE 8 CHARS
  000AF6 0AD9                         3542+         SVC   C'R' .                       READ IT
  000AF8 D503 137C 200C 00E44 0000C   3543+         CLC   =C'READ',XRXTEXT .           IF 1ST WORD IS READ, OK
  000AFE 4770 1024           00AEC    3544+         BNE   RDRHLOOP .                   ELSE IGNORE
  000B02 5850 2010           00010    3545+         L     5,XRXTEXT+4 .                GET 2ND WORD OF THE TEXT
                                      3546+         DROP  2
  000B06 4120 3004           00004    3547+         LA    2,UCBUS .                    LOCK THE UCB AND ITS UNIT
  000B0A 0AD7                         3548+         SVC   C'P'
  000B0C 4120 C008           00008    3549+         LA    2,RDRHMSG .                  RESET ADDRESSING POINTER
  000000                              3550+         USING XRX,2
  000B10 95FF C07A     0007A          3551+         CLI   JOBBIT,X'FF' .               HAVE WE JUST READ $JOB CARD?
  000B14 4770 1066           00B2E    3552+         BNE   RDRHMORE .                   IF NO, GO CHECK PROTECTION. ELSE
  000B18 955C 2000     00000          3553+         CLI   XRXNAME,C'*' .               IS JSP CALLING US?
  000B1C 4770 10D8           00BA0    3554+         BNE   RDRHNO .                     IF NOT, TELL HIM NO.
  000B20 D24F 5000 C01C 00000 0001C   3555+         MVC   0(80,5),RDRHTEMP .           IF IT IS, GIVE JSP THE $JOB CARD
  000B26 9200 C07A     0007A          3556+         MVI   JOBBIT,X'00' .               SAY WE DON'T HAVE $JOB WAITING
  000B2A 47F0 10F4           00BBC    3557+         B     RDRHSOK .                     AND SEND MESSAGE BACK
                                      3558+         DROP  2
  000B2E 955C C008     00008          3559+RDRHMORE CLI   RDRHMSG,C'*' .               IS SYSTEM CALLING?
  000B32 4780 108C           00B54    3560+         BE    RDRHPOK .                    THEN PROTECTION OK. ELSE
  000B36 18B5                         3561+         LR    11,5 .                       GET ADDRESS THAT'S TO HOLD CARD,'
  000B38 54B0 114C           00C14    3562+         N     11,PROTCON1 .                GET THE 2K BOUNDARY
  000B3C 09AB                         3563+         ISK   10,11 .                      FIND STORAGE KEY
```

```
          SAMPLE OPERATING SYSTEM      VERSION 2.00

  LOC   OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT

000B3E 19A4                          3564+           CR    10,4 .                                    DOES IT MATCH OURS?
000B40 4770 10D8              00BA0  3565+           BNE   RDRHNO .                                  IF NOT, TELL HIM NO
000B44 41B5 004F              0004F  3566+           LA    11,79(5) .                                CHECK LAST BYTE ADDR OF CARD
000B48 54B0 114C              00C14  3567+           N     11,PROTCON1 .                             GET THE 2K BOUNDARY
000B4C 09AB                          3568+           ISK   10,11 .                                   FIND STORAGE KEY
000B4E 19A4                          3569+           CR    10,4 .                                    DOES IT MATCH OURS?
000B50 4770 10D8              00BA0  3570+           BNE   RDRHNO .                                  IF NOT, TELL HIM NO
000B54 5450 1148              00C10  3571+RDRHPOK    N     5,CCBCON1 .                               MAKE ADDRESS INTO
000B58 5050 C000              00000  3572+           ST    5,RDRHCCB .                               A CCW (OR CCB)
000B5C 9602 C000        00000        3573+           OI    RDRHCCB,X'02'
000B60 D203 C004 1380 00004 00E48    3574+           MVC   RDRHCCB+4,=F'80' .                        WE'LL READ EIGHTY CHARACTERS
000B66 D203 3014 1368 00014 00E30    3575+           MVC   UCBCSW(4),=A(0) .                         CLEAR THE LAST CSW THERE
000B6C D203 3018 1368 00018 00E30    3576+           MVC   UCBCSW+4(4),=A(0)
000B72 4120 0194              00194  3577+           LA    2,CAWSEM .                                LOCK THE CAW
000B76 0AD7                          3578+           SVC   C'P'
000B78 5060 0048              00048  3579+           ST    6,CAW .                                   THAT'S THE CAW
000B7C 5870 3000              00000  3580+           L     7,UCBADDR .                               GET THE UNIT ADDRESS
000B80 9C00 7000        00000        3581+           SIO   0(7) .                                    START THE I/O
000B84 4770 1134              00BFC  3582+           BNZ   RDSTATUS .                                BRANCH IF SIO UNSUCCESSFUL
000B88 0AE5                          3583+           SVC   C'V' .                                    THEN UNLOCK THE CAW
000B8A 4120 300C              0000C  3584+RDRHWAIT   LA    2,UCBWS .                                 NOW WAIT FOR AN INTERRUPT
000B8E 0AD7                          3585+           SVC   C'P'
000B90 9105 3018        00018        3586+           TM    UCBCSW+4,X'05' .                          CHECK THE STATUS
000B94 4780 10C2              00B8A  3587+           BZ    RDRHWAIT .                                IF NOT FINISHED, WAIT
000B98 9101 3018        00018        3588+           TM    UCBCSW+4,X'01' .                          CHECK FOR EXCEPTION
000B9C 4780 10E2              00BAA  3589+           BZ    RDRHOK .                                  IF NOT, ALL IS GROOVY
000BA0 D201 C078 139C 00078 00E64    3590+RDRHNO     MVC   RDRHM+12(2),=C'NO' .                      ELSE MESSAGE BACK IS NO
000BA6 47F0 10FA              00BC2  3591+           B     RDRHSEND .                                GET READY TO SEND
000BAA 955C C008        00008        3592+RDRHOK     CLI   RDRHMSG,C'*' .                            IS THE SYSTEM CALLING?
000BAE 4780 10F4              00BBC  3593+           BE    RDRHSOK .                                 THAT'S FINE. OTHERWISE,
000BB2 D504 13A0 5000 00E68 00000    3594+           CLC   =C'$JOB,',0(5) .                          WAS IT A $JOB CARD?
000BB8 4780 1116              00BDE  3595+           BE    ENDADATA .                                OOPS! WE HIT END OF DATA STREAM
000BBC D201 C078 139E 00078 00E66    3596+RDRHSOK    MVC   RDRHM+12(2),=C'OK' .                      GROOVINESS MESSAGE
000BC2 D203 C074 1378 00074 00E40    3597+RDRHSEND   MVC   RDRHM+8(4),=F'2' .                        SAY THERE ARE 2 CHARACTERS
000BC8 D207 C06C C008 0006C 00008    3598+           MVC   RDRHM+0(8),RDRHMSG+0 .                    SEND BACK TO SAME GUY
000BCE 4120 3004              00004  3599+           LA    2,UCBUS .                                 NOW UNLOCK UCB AND UNIT
000BD2 0AE5                          3600+           SVC   C'V'
000BD4 4120 C06C              0006C  3601+           LA    2,RDRHM .                                 SET UP MESSAGE
000BD8 0AE2                          3602+           SVC   C'S' .                                    AND SEND IT
000BDA 47F0 1024              00AEC  3603+           B     RDRHLOOP
000BDE D201 C078 139C 00078 00E64    3604+ENDADATA   MVC   RDRHM+12(2),=C'NO' .                      TELL USER NO MORE CARDS
000BE4 D24F C01C 5000 0001C 00000    3605+           MVC   RDRHTEMP(80),0(5) .                       SAVE THE $JOB CARD
000BEA 9240 5000        00000        3606+           MVI   0(5),C' ' .                               BLANK OUT THE USER'S COPY
000BEE D24E 5001 5000 00001 00000    3607+           MVC   1(79,5),0(5)
000BF4 92FF C07A        0007A        3608+           MVI   JOBBIT,X'FF' .                            INDICATE WE HAVE A NEW $JOB CARD
000BF8 47F0 10FA              00BC2  3609+           B     RDRHSEND .                                AND SEND MESSAGE BACK
000BFC 0AE5                          3610+RDSTATUS   SVC   C'V' .                                    UNLOCK THE CAW
000BFE 4120 300C              0000C  3611+           LA    2,UCBWS .                                 AND WAIT FOR AN INTERRUPT
000C02 0AD7                          3612+           SVC   C'P'
000C04 47F0 108C              00B54  3613+           B     RDRHPOK .                                 AND TRY TO RESTART THE I/O
                                     3614+           DROP  3,12

000C08 0000000100000000              3616+RDRHSEM    DC    F'1,0'
000C10 00FFFFFF                      3617+CCBCON1    DC    X'00FFFFFF' MASK
000C14 00FFF800                      3618+PROTCON1   DC    X'00FFF800'
```

```
          SAMPLE OPERATING SYSTEM      VERSION 2.00
   LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT

000C18 00000080                      3619+RDRHAAS   DC    A(LENRDRHA) ALLOCATE ARGLIST FOR STORAGE
000C1C 00000000                      3620+          DC    F'0'
000C20 00000008                      3621+          DC    F'8'


                                     3623+***********************************************************************
                                     3624+*                                                                     *
                                     3625+*              SYSTEM SUPPLIED DEVICE HANDLER FOR PRINTERS            *
                                     3626+*                                                                     *
                                     3627+***********************************************************************
000C24                               3629+PRTHANDL  EQU   * .                    THE PRINTER HANDLER
000000                               3630+          USING UCB,3 .                ENTERED WITH REG3 -> THE UCB
000C24 0510                          3631+          BALR  1,0
000C26                               3632+          USING *,1 .                  ESTABLISH ADDRESSING
000C26 4120 110A           00D30     3633+          LA    2,PRTHSEM .            LOCK UNTIL ALLOCATE STORAGE
000C2A 0AD7                          3634+          SVC   C'P'
000C2C 4120 1112           00D38     3635+          LA    2,PRTHAAS .            READY TO ALLOCATE
000000                               3636+          USING XAX,2
000C30 0AC5                          3637+          SVC   C'E' .                 ALLOCATE
000C32 58C0 2004           00004     3638+          L     12,XAXADDR .           GET THE ADDRESS
                                     3639+          DROP  2
000C36 4120 110A           00D30     3640+          LA    2,PRTHSEM
000C3A 0AE5                          3641+          SVC   C'V' .                 UNLOCK TO ROUTINE
000C3C 8840 0010           00010     3642+          SRL   4,16 .                 SHIFT KEY
000C40 1BAA                          3643+          SR    10,10 .                CLEAR REG 10
000000                               3644+          USING PRTHAS,12 .            ADDRESSING IN THE AUTO AREA
000C42 4160 C000           00000     3645+          LA    6,PRTHCCB .            MAKE A CAW
000C46 4120 C008           00008     3646+PRTHLOOP  LA    2,PRTHMSG .            READY TO READ A MESSAGE
000000                               3647+          USING XRX,2
000C4A D203 2008 1212 00008 00E38    3648+          MVC   XRXSIZE,=F'8' .        WE CAN TAKE 8 CHARACTERS
000C50 0AD9                          3649+          SVC   C'R' .                 READ IT
000C52 5850 2010           00010     3650+          L     5,XRXTEXT+4 .          LOAD THE ADDRESS
000C56 D503 1226 200C 00E4C 0000C    3651+          CLC   =C'PRIN',XRXTEXT .     IS IT A PRIN REQUEST?
000C5C 4780 1048           00C6E     3652+          BE    PRTHPRIN
000C60 D503 122A 200C 00E50 0000C    3653+          CLC   =C'STC1',XRXTEXT .     OR A SKIP REQUEST?
000C66 4780 108A           00CB0     3654+          BE    PRTHSTC1
000C6A 47F0 1020           00C46     3655+          B     PRTHLOOP .             IF NEITHER, IGNORE
                                     3656+          DROP  2
000C6E 4120 3004           00004     3657+PRTHPRIN  LA    2,UCBUS
000C72 0AD7                          3658+          SVC   C'P' .                 LOCK THE UCB AND UNIT
000C74 955C C008     00008           3659+          CLI   PRTHMSG,C'*' .         IS SYSTEM CALLING?
000C78 4780 1074           00C9A     3660+          BE    PRTHPOK .              THEN PROTECTION OK. ELSE
000C7C 18B5                          3661+          LR    11,5 .                 GET ADDRESS THAT'S TO HOLD MSG,
000C7E 54B0 0C14           00C14     3662+          N     11,PROTCON1 .          GET THE 2K BOUNDARY
000C82 09AB                          3663+          ISK   10,11 .                FIND STORAGE KEY
000C84 19A4                          3664+          CR    10,4 .                 DOES IT MATCH OURS?
000C86 4770 10D0           00CF6     3665+          BNE   PRTHNO .               IF NOT, TELL HIM NO
000C8A 41B5 0083           00083     3666+          LA    11,131(5) .            CHECK LAST BYTE ADDRESS OF LINE
000C8E 54B0 0C14           00C14     3667+          N     11,PROTCON1 .          GET THE 2K BOUNDARY
000C92 09AB                          3668+          ISK   10,11 .                FIND STORAGE KEY
000C94 19A4                          3669+          CR    10,4 .                 DOES IT MATCH OURS?
000C96 4770 10D0           00CF6     3670+          BNE   PRTHNO .               IF NOT, TELL HIM NO
000C9A 5450 0C10           00C10     3671+PRTHPOK   N     5,CCBCON1 .            MAKE A WRITE REQUEST
```

```
          SAMPLE OPERATING SYSTEM     VERSION 2.00

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

000C9E 5050 C000            00000  3672+         ST    5,PRTHCCB .             FOR THE CCB
000CA2 9609 C000      00000        3673+         OI    PRTHCCB,X'09' .         PRINT COMMAND CODE
000CA6 D203 C004 122E 00004 00E54  3674+         MVC   PRTHCCB+4,=F'132' .     WE'LL PRINT 132 CHARACTERS
000CAC 47F0 1096            00CBC  3675+         B     PRTHCOMM .              BRANCH TO COMMON SECTION
000CB0 D207 C000 11FA 00000 00E20  3676+PRTHSTC1 MVC   PRTHCCB(8),=X'8900000020000001' SKIP TO TOP OF PAGE
000CB6 4120 3004            00004  3677+         LA    2,UCBUS
000CBA 0AD7                        3678+         SVC   C'P' .                  LOCK THE UCB AND UNIT
000CBC 4120 0194            00194  3679+PRTHCOMM LA    2,CAWSEM .              LOCK THE CAW
000CC0 0AD7                        3680+         SVC   C'P'
000CC2 5060 0048            00048  3681+         ST    6,CAW .                 STORE OUR CAW
000CC6 D203 3014 120A 00014 00E30  3682+         MVC   UCBCSW(4),=A(0) .       CLEAR THE LAST CSW THERE
000CCC D203 3018 120A 00018 00E30  3683+         MVC   UCBCSW+4(4),=A(0)
000CD2 5870 3000            00000  3684+         L     7,UCBADDR .             GET THE ADDRESS
000CD6 9C00 7000      00000        3685+         SIO   0(7) .                  START THE I/O
000CDA 4770 10FC            00D22  3686+         BNZ   PTSTATUS .              BRANCH IF SIO UNSUCCESSFUL
000CDE 0AE5                        3687+         SVC   C'V' .                  AND UNLOCK THE CAW
000CE0 4120 300C            0000C  3688+PRTHWAIT LA    2,UCBWS .               START TO WAIT
000CE4 0AD7                        3689+         SVC   C'P'
000CE6 9105 3018      00018        3690+         TM    UCBCSW+4,X'05' .        IS THE UNIT READY?
000CEA 4780 10BA            00CE0  3691+         BZ    PRTHWAIT .              IF NOT, ITS STILL ON. WAIT
000CEE 9101 3018      00018        3692+         TM    UCBCSW+4,X'01' .        WAS THERE AN EXCEPTION?
000CF2 4780 10DA            00D00  3693+         BZ    PRTHOK .                IF NOT, GOOD
000CF6 D201 C028 123E 00028 00E64  3694+PRTHNO   MVC   PRTHM+12(2),=C'NO' .    THERE WAS, SO SAY SO
000CFC 47F0 10E0            00D06  3695+         B     PRTHSEND
000D00 D201 C028 1240 00028 00E66  3696+PRTHOK   MVC   PRTHM+12(2),=C'OK' .    NO ERRORS
000D06 D203 C024 121A 00024 00E40  3697+PRTHSEND MVC   PRTHM+8(4),=F'2' .      SENDING 2 CHARACTERS
000D0C D207 C01C C008 0001C 00008  3698+         MVC   PRTHM+0(8),PRTHMSG+0 .  SEND TO OUR SENDER
000D12 4120 3004            00004  3699+         LA    2,UCBUS
000D16 0AE5                        3700+         SVC   C'V' .                  UNLOCK THE UCB
000D18 4120 C01C            0001C  3701+         LA    2,PRTHM
000D1C 0AE2                        3702+         SVC   C'S' .                  SEND IT
000D1E 47F0 1020            00C46  3703+         B     PRTHLOOP .              AND READ ANOTHER MESSAGE
000D22 0AE5                        3704+PTSTATUS SVC   C'V' .                  UNLOCK THE CAW
000D24 4120 300C            0000C  3705+         LA    2,UCBWS .               AND WAIT FOR AN INTERRUPT
000D28 0AD7                        3706+         SVC   C'P'
000D2A 47F0 1096            00CBC  3707+         B     PRTHCOMM .              AND TRY TO RESTART THE I/O
                                   3708+         DROP  3,12

000D2E 0000
000D30 0000000100000000            3710+PRTHSEM  DC    F'1,0' LOCK
000D38 00000030                    3711+PRTHAAS  DC    A(LENPRTHA) XA ARG LIST FOR AUTO STORAGE
000D3C 00000000                    3712+         DC    F'0'
000D40 00000008                    3713+         DC    F'8'
```

```
              SAMPLE OPERATING SYSTEM |    VERSION 2.00
       LOC  OBJECT CODE    ADDR1 ADDR2  STMT    SOURCE STATEMENT
                                        3715+*********************************************************************
                                        3716+*                                                                   *
                                        3717+*         SYSTEM ROUTINE FOR USER SUPPLIED DEVICE HANDLER           *
                                        3718+*                                                                   *
                                        3719+*********************************************************************

      000D44                            3721+EXCPHNDL EQU   * .                      EXCP DEVICE HANDLER
      000000                            3722+         USING UCB,3 .                  WILL HAVE REG 3 -> UCB
      000D44 0510                       3723+         BALR  1,0
      000D46                            3724+         USING *,1 .                    ESTABLISH ADDRESSING
      000D46 4120 10C6            00E0C 3725+         LA    2,EXCPHSEM .             LOCK OURSELVES UNTIL WE HAVE
      000D4A 0AD7                       3726+         SVC   C'P' .                   SET UP AUTOMATIC STORAGE
      000D4C 4120 10CE            00E14 3727+         LA    2,EXCPHAAS .             READY TO ALLOCATE
      000000                            3728+         USING XAX,2
      000D50 0AC5                       3729+         SVC   C'E' .                   ALLOCATE
      000D52 58C0 2004            00004 3730+         L     12,XAXADDR .             GET POINTER TO AUTO STORAGE
                                        3731+         DROP  2
      000D56 4120 10C6            00E0C 3732+         LA    2,EXCPHSEM .             AND UNLOCK OURSELVES
      000D5A 0AE5                       3733+         SVC   C'V'
      000D5C 184B                       3734+         LR    4,11
      000D5E 8940 0008            00008 3735+         SLL   4,8 .                    SHIFT KEY FOR CAW
      000000                            3736+         USING EXCPHAS,12 .             FOR ADDRESSING AUTO AREA
      000D62 4120 C000            00000 3737+EXCPLOOP LA    2,EXCPHMSG .             TRY TO READ A MESSAGE
      000000                            3738+         USING XRX,2
      000D66 D203 2008 1112 00008 00E58 3739+         MVC   XRXSIZE,=F'12' .         WE'LL TAKE 12 CHARACTERS
      000D6C 0AD9                       3740+         SVC   C'R'
      000D6E D503 1116 200C 00E5C 0000C 3741+         CLC   =C'EXCP',XRXTEXT .       IS IT AN EXCP MESSAGE?
      000D74 4770 101C            00D62 3742+         BNE   EXCPLOOP .               IF NOT, IGNORE IT
      000D78 5850 2010            00010 3743+         L     5,XRXTEXT+4 .            REG 5 CONTAINS CHAN AND DEV
      000D7C 5860 2014            00014 3744+         L     6,XRXTEXT+8 .            REG 6 CONTAINS ADDR OF CCWS
                                        3745+         DROP  2
      000D80 4170 112E            00E74 3746+         LA    7,UCBTABLE .             GET PTR TO UCB TABLE
      000D84 5957 0000            00000 3747+EXCPCOMP C     5,0(7) .                 COMPARE UNIT ADDRESS
      000D88 4780 1054            00D9A 3748+         BE    EXCPFIND .               THAT'S THE UCB WE WANT
      000D8C 4177 0020            00020 3749+         LA    7,UCBLENG(7) .           GET PTR TO NEXT UCB
      000D90 5970 111A            00E60 3750+         C     7,=A(UCBTBEND) .         ARE WE THROUGH WITH TABLE?
      000D94 4770 103E            00D84 3751+         BNE   EXCPCOMP .               IF NOT, LOOK SOME MORE
      000D98 0A6F                       3752+         SVC   C'?' .                   ELSE ERROR
      000D9A 1837                       3753+EXCPFIND LR    3,7 .                    SET REG 3 TO UCB PTR
      000D9C 4120 3004            00004 3754+         LA    2,UCBUS
      000DA0 0AD7                       3755+         SVC   C'P' .                   LOCK THE UCB
      000DA2 1664                       3756+         OR    6,4 .                    OR IN USER'S KEY
      000DA4 D203 3014 10EA 00014 00E30 3757+         MVC   UCBCSW(4),=A(0) .        CLEAR THE LAST CSW THERE
      000DAA D203 3018 10EA 00018 00E30 3758+         MVC   UCBCSW+4(4),=A(0)
      000DB0 4120 0194            00194 3759+         LA    2,CAWSEM
      000DB4 0AD7                       3760+         SVC   C'P' .                   LOCK CAW
      000DB6 5060 0048            00048 3761+         ST    6,CAW .                  STORE OUR CAW
      000DBA 9C00 5000      00000       3762+         SIO   0(5) .                   START THE I/O
      000DBE 0AE5                       3763+         SVC   C'V' .                   UNLOCK CAW
      000DC0 4120 300C            0000C 3764+EXCPWAIT LA    2,UCBWS .                NOW WAIT FOR AN INTERRUPT
      000DC4 0AD7                       3765+         SVC   C'P'
      000DC6 D207 C024 3014 00024 00014 3766+         MVC   EXCPHM+12(8),UCBCSW .    GIVE USER HIS CSW
      000DCC D203 C020 1112 00020 00E58 3767+         MVC   EXCPHM+8(4),=F'12'
      000DD2 D207 C018 C000 00018 00000 3768+         MVC   EXCPHM(8),EXCPHMSG
      000DD8 4120 C018            00018 3769+         LA    2,EXCPHM
```

```
       SAMPLE OPERATING SYSTEM I    VERSION 2.00

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

000DDC 0AE2                        3770+           SVC   C'S' .                    AND SEND THE MESSAGE
000DDE 4120 C000            00000  3771+           LA    2,EXCPHMSG .              AND WAIT FOR A REPLY
000000                             3772+           USING XRX,2
000DE2 D203 2008 10F2 00008 00E38  3773+           MVC   XRXSIZE(4),=F'8' .        FROM THE USER
000DE8 0AD9                        3774+           SVC   C'R'
000DEA D501 1120 200C 00E66 0000C  3775+           CLC   =C'OK',XRXTEXT .          AM I DONE?
000DF0 4780 10BA            00E00  3776+           BE    EXCPDONE
000DF4 D504 1127 200C 00E6D 0000C  3777+           CLC   =C'AGAIN',XRXTEXT .       DOES HE WANT ANOTHER CSW?
000DFA 4780 107A            00DC0  3778+           BE    EXCPWAIT
000DFE 0A6F                        3779+           SVC   C'?' .                    WRONG MESSAGE
                                   3780+           DROP  2
000E00 4120 3004            00004  3781+EXCPDONE LA    2,UCBUS .                 UNLOCK UNIT
000E04 0AE5                        3782+           SVC   C'V'
000E06 47F0 101C            00D62  3783+           B     EXCPLOOP .                AND GET ANOTHER MESSAGE
                                   3784+           DROP  3,12
000E0A 0000
000E0C 0000000100000000            3785+EXCPHSEM DC    F'1,0'
000E14 00000030                    3786+EXCPHAAS DC    A(LENEXCPA) .             ALLOCATION OF AUTO STORAGE
000E18 00000000                    3787+           DC    F'0'
000E1C 00000008                    3788+           DC    F'8'


000E20                             3790+           LTORG
000E20 8900000020000001            3791                  =X'8900000020000001'
000E28 00000001                    3792                  =F'1'
000E2C 00000600                    3793                  =A(XA)
000E30 00000000                    3794                  =A(0)
000E34 00000148                    3795                  =A(LENPCB)
000E38 00000008                    3796                  =F'8'
000E3C FFFFFFF8                    3797                  =F'-8'
000E40 00000002                    3798                  =F'2'
000E44 D9C5C1C4                    3799                  =C'READ'
000E48 00000050                    3800                  =F'80'
000E4C D7D9C9D5                    3801                  =C'PRIN'
000E50 E2E3C3F1                    3802                  =C'STC1'
000E54 00000084                    3803                  =F'132'
000E58 0000000C                    3804                  =F'12'
000E5C C5E7C3D7                    3805                  =C'EXCP'
000E60 00000EF4                    3806                  =A(UCBTBEND)
000E64 D5D6                        3807                  =C'NO'
000E66 D6D2                        3808                  =C'OK'
000E68 5BD1D6C26B                  3809                  =C'$JOB,'
000E6D C1C7C1C9D5                  3810                  =C'AGAIN'
```

```
         SAMPLE OPERATING SYSTEM I     VERSION 2.00

  LOC   OBJECT CODE      ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                      3812+***********************************************************************
                                      3813+*                                                                     *
                                      3814+*                         UNIT CONTROL BLOCKS                         *
                                      3815+*                                                                     *
                                      3816+***********************************************************************
000E74                                3818+UCBTABLE DS    0F .                TABLE OF UNIT CONTROL BLOCKS
                                      3819              *,            UCB FOR READER 1
000E74 00000012                       3820+UCBRDR1  DC    X'00000012' .       DEVICE ADDRESS,
000E78 0000000100000000               3821+         DC    F'1,0' .            USER SEMAPHORE,
000E80 0000000000000000               3822+         DC    F'0,0' .            WAIT SEMAPHORE,
000E88 0000000000000000               3823+         DC    F'0,0' .            CHANNEL STATUS WORD
000E90 00                             3824+         DC    X'00'
000E94                                3825+         DS    0F
                                      3826              *,            UCB FOR PRINTER 1
000E94 00000010                       3827+UCBPRT1  DC    X'00000010' .       DEVICE ADDRESS,
000E98 0000000100000000               3828+         DC    F'1,0' .            USER SEMAPHORE,
000EA0 0000000000000000               3829+         DC    F'0,0' .            WAIT SEMAPHORE,
000EA8 0000000000000000               3830+         DC    F'0,0' .            CHANNEL STATUS WORD
000EB0 00                             3831+         DC    X'00'
000EB4                                3832+         DS    0F
                                      3833              *,            UCB FOR READER 2
000EB4 0000000C                       3834+UCBRDR2  DC    X'0000000C' .       DEVICE ADDRESS,
000EB8 0000000100000000               3835+         DC    F'1,0' .            USER SEMAPHORE,
000EC0 0000000000000000               3836+         DC    F'0,0' .            WAIT SEMAPHORE,
000EC8 0000000000000000               3837+         DC    F'0,0' .            CHANNEL STATUS WORD
000ED0 00                             3838+         DC    X'00'
000ED4                                3839+         DS    0F
                                      3840              *,            UCB FOR PRINTER 2
000ED4 0000000E                       3841+UCBPRT2  DC    X'0000000E' .       DEVICE ADDRESS,
000ED8 0000000100000000               3842+         DC    F'1,0' .            USER SEMAPHORE,
000EE0 0000000000000000               3843+         DC    F'0,0' .            WAIT SEMAPHORE,
000EE8 0000000000000000               3844+         DC    F'0,0' .            CHANNEL STATUS WORD
000EF0 00                             3845+         DC    X'00'
000EF4                                3846+         DS    0F
000EF4                                3847+UCBTBEND EQU   *
```

```
            SAMPLE OPERATING SYSTEM!      VERSION 2.00

   LOC   OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                     3849+***********************************************************************
                                     3850+*                                                                     *
                                     3851+*                        I/O INTERRUPT HANDLER                        *
                                     3852+*                                                                     *
                                     3853+***********************************************************************

  000EF4                             3855+IOHANDL  EQU   * .                       THE I/O INTERRUPT HANDLER
  000EF4 900F 01DC            001DC  3856+         STM   0,15,IOHSAVE .            SAVE REGISTERS
  000EF8 0510                        3857+         BALR  1,0
  000EFA                             3858+         USING *,1 .                   ESTABLISH ADDRESSING
  000EFA 94FD 0039      00039        3859+         NI    IOOLD+1,X'FD' .         TURN OFF WAIT BIT
  000EFE 5860 16FE            015F8  3860+         L     6,=A(UCBTABLE) .        GET POINTER TO UCB TABLE
  000F02 D501 6002 003A 00002 0003A  3861+IOCOMP   CLC   2(2,6),IOOLD+2 .        COMPARE DEVICE AND CHANNEL
  000F08 4780 1022            00F1C  3862+         BE    IODEVFND .              IF EQUAL, REG 6 INDICATES PTR
  000F0C 4166 0020            00020  3863+         LA    6,UCBLENG(6) .          INCREMENT TO NEXT ENTRY
  000F10 5960 1702            015FC  3864+         C     6,=A(UCBTBEND) .        ARE WE AT END OF TABLE?
  000F14 4770 1008            00F02  3865+         BNE   IOCOMP .                IF NOT DONE, TRY NEXT UCB
  000F18 47F0 106C            00F66  3866+         B     IOBACK .                ELSE, IGNORE IT
  000000                             3867+         USING UCB,6 .                 IT'S A UCB PTR
  000F1C D203 6014 0040 00014 00040  3868+IODEVFND MVC   UCBCSW(4),CSW .         MOVE IN THE NEW CSW
  000F22 5870 0044            00044  3869+         L     7,CSW+4 .               GET STATUS BYTE
  000F26 5670 6018            00018  3870+         O     7,UCBCSW+4 .            OR IN NEW STATUS INFORMATION
  000F2A 5070 6018            00018  3871+         ST    7,UCBCSW+4 .             AND STORE IT BACK
  000F2E D201 601A 0046 0001A 00046  3872+         MVC   UCBCSW+6(2),CSW+6 .     MOVE IN BYTE COUNT
  000F34 4120 600C            0000C  3873+         LA    2,UCBWS
  000F38 9500 601C      0001C        3874+         CLI   UCBFPR,X'00' .          IS FAST PROCESSING
  000F3C 4780 106A            00F64  3875+         BE    IONOFPR .               REQUIRED? IF NOT, RETURN
  000F40 58F0 0270            00270  3876+         L     15,RUNNING .            IF SO, STOP GUY NOW RUNNING
  000000                             3877+         USING PCB,15
  000F44 95FF F019      00019        3878+         CLI   PCBBLOKT,X'FF' .        IS ANYONE REALLY RUNNING?
  000F48 4780 1062            00F5C  3879+         BE    IOWAIT .                IF NOT, START UP SLEEPER
  000F4C 41D0 F04C            0004C  3880+         LA    13,PCBISA .             IF SO, STOP RUNNING PROCESS
  000000                             3881+         USING SA,13
  000F50 D207 D000 0038 00000 00038  3882+         MVC   SAPSW,IOOLD .           SAVE PROCCESS WHICH WAS
  000F56 D23F D008 01DC 00008 001DC  3883+         MVC   SAREGS,IOHSAVE .         INTERRUPTED
                                     3884+         DROP  13,15
  000F5C 9200 0278      00278        3885+IOWAIT   MVI   NEXTTRYM,X'00' .        MAKE NEXTTRY NOT MODIFIED
  000F60 0AE5                        3886+         SVC   C'V' .                  SO CAN FAST PROCCESS SLEEPER
  000F62 0A4B                        3887+         SVC   C'.' .                  GO PROCESS IT RIGHT AWAY
  000F64 0AE5                        3888+IONOFPR  SVC   C'V' .                  AND WAKE UP THE SLEEPER
  000F66 980F 01DC            001DC  3889+IOBACK   LM    0,15,IOHSAVE .          RELOAD OUR REGISTERS
  000F6A 8200 0038      00038        3890+         LPSW  IOOLD .                 AND STEALTHILY RETURN
                                     3891+         DROP  1,6
```

```
          SAMPLE OPERATING SYSTEM I     VERSION 2.00

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                   3893+**********************************************************************
                                   3894+*                                                                    *
                                   3895+*                          IPL ENTERED ROUTINE                       *
                                   3896+*                                                                    *
                                   3897+*        FUNCTION: TO INITIALIZE SYSTEM PARAMETERS, SET STORAGE KEYS, *
                                   3898+*                  AND CREATE MULTIPLE JOB STREAMS.                  *
                                   3899+*                                                                    *
                                   3900+**********************************************************************

000F6E                             3902+IPLRTN   EQU   * .                            THE IPL-ENTERED ROUTINE
000F6E 0510                        3903+         BALR  1,0
000F70                             3904+         USING *,1 .                          ESTABLISH ADDRESSING
000F70 41F0 1098            01008  3905+         LA    15,IPLPCB .                    I'M RUNNING
000F74 50F0 0270            00270  3906+         ST    15,RUNNING .                   INITIALIZE 'RUNNING'
000F78 50F0 0274            00274  3907+         ST    15,NEXTTRY .                   INITIALIZE 'NEXTTRY'
000F7C D207 16D8 1660 01648 015D0  3908+         MVC   VERYEND,=A(0,CORESIZE-(VERYEND-PROGRAM)) FREE CORE
000F82 4130 0008            00008  3909+         LA    3,8 .                          SET ZERO KEY AND FETCH PROTECT
000F86 5820 11EC            0115C  3910+         L     2,CORESIZ .                    START PAST THE LAST BLOCK
000F8A 5B20 1690            01600  3911+IPLCL    S     2,=F'2048' .                   GO TO THE PREVIOUS BLOCK
000F8E 4740 1028            00F98  3912+         BM    IPLTH .                        IF NEGATIVE, WE'RE THROUGH HERE
000F92 0832                        3913+         SSK   3,2 .                          ELSE SET THE STORAGE KEY TO
000F94 47F0 101A            00F8A  3914+         B     IPLCL .                         ZERO AND WORK BACKWARDS
000F98 1B44                        3915+IPLTH    SR    4,4 .                          INDEX IN TABLES FOR INPUT STREAM
000F9A 5850 1074            00FE4  3916+         L     5,STREAMS .                    HOW MANY STREAMS?
000F9E 4120 11E0            01150  3917+IPLLOOP  LA    2,IPLAPCBS .                   READY TO ALLOCATE A PCB
000000                             3918+         USING XAX,2
000FA2 0AC1                        3919+         SVC   C'A' .                         ALLOCATE
000FA4 5820 2004            00004  3920+         L     2,XAXADDR .                    GET THE ADDRESS
000FA8 D253 2000 11F0 00000 01160  3921+         MVC   0(TYPLEN,2),TYPPCB .           MAKE IT LOOK LIKE A PCB
000FAE 0AC9                        3922+         SVC   C'I' .                         CHAIN IT ON
000000                             3923+         USING PCB,2
000FB0 5020 2008            00008  3924+         ST    2,PCBNPTG .                    BUT PUT IT IN A GROUP BY ITSELF
000FB4 5020 200C            0000C  3925+         ST    2,PCBLPTG
                                   3926+         DROP  2
000000                             3927+         USING PCB,15
000FB8 50F0 F00C            0000C  3928+         ST    15,PCBLPTG .                   LIKEWISE FOR THE IPL PCB
000FBC 50F0 F008            00008  3929+         ST    15,PCBNPTG
                                   3930+         DROP  15
000000                             3931+         USING PCB,2
000FC0 4180 204C            0004C  3932+         LA    8,PCBISA .                     GET THE NEW PCB'S ISA
000000                             3933+         USING SA,8
000FC4 4190 8008            00008  3934+         LA    9,SAREGS .                     ABOUT TO FIX INIT REGS
000000                             3935+         USING REGS,9
000FC8 41A0 1078            00FE8  3936+         LA    10,UCBTAB
000FCC 1AA4                        3937+         AR    10,4
000FCE D203 900C A000 0000C 00000  3938+         MVC   REG3,0(10) .                   REG3 -> (RDRUCB,PTRUCB)
000FD4 D203 9010 A008 00010 00008  3939+         MVC   REG4,KEYTAB-UCBTAB(10)  .REG4 = KEY
                                   3940+         DROP  9
000FDA 4144 0004            00004  3941+         LA    4,4(4) .                       GO TO NEXT JOB STREAM
000FDE 4650 102E            00F9E  3942+         BCT   5,IPLLOOP .                    DO FOR EACH STREAM
000FE2 0A4B                        3943+         SVC   C'.' .                         THEN ENTER TRAFFIC CONTROLLER

000FE4 00000002                    3945+STREAMS  DC    F'2' .                         NUMBER OF STREAMS
```

```
                SAMPLE OPERATING SYSTEM1    VERSION 2.00

  LOC  OBJECT CODE     ADDR1 ADDR2  STMT     SOURCE STATEMENT

000FE8                             3947+UCBTAB   EQU    *.                      TABLE OF PTRS TO UCB BLOCKS
000FE8 00000FF8                    3948+         DC     A(UCBLP1)
000FEC 00001000                    3949+         DC     A(UCBLP2)

000FF0                             3951+KEYTAB   EQU    *.                      TABLE OF PROTECTION KEYS
000FF0 00100000                    3952+         DC     X'00100000'
000FF4 00200000                    3953+         DC     X'00200000'

000FF8 00000E7400000E94            3955+UCBLP1   DC     A(UCBRDR1,UCBPRT1)
001000 00000EB400000ED4            3956+UCBLP2   DC     A(UCBRDR2,UCBPRT2)

001008                             3958+         DS     0D
001008 4040404040404040            3959+IPLPCB   DC     CL8' '.                 IPL ROUTINE PCB
001010 0000100800001008            3960+         DC     4A(IPLPCB)
001020 FF000000                    3961+         DC     X'FF000000'.            INITIALIZED FLAGS
001024 0000000100000000            3962+         DC     F'1,0'
00102C 0000000000000000            3963+         DC     5F'0,0'
001054 0002000000000000            3964+         DC     X'0002000000000000'
00105C                             3965+         DS     CL76
0010A8                             3966+         DS     CL84
0010FC                             3967+         DS     CL84

001150 00000148                    3969+IPLAPCBS DC     A(LENPCB).              ALLOC LIST FPR PCB'S
001154 00000000                    3970+         DC     A(0)
001158 00000008                    3971+         DC     F'8'
00115C 00008000                    3972+CORESIZ  DC     A(CORESIZE).            BYTES OF CORE IN OBJECT MACHINE

001160                             3974+         DS     0D
001160 5CC9C2E2E4D74040            3975+TYPPCB   DC     CL8'*IBSUP'.            A TEMPLATE *IBSUP PCB
001168 0000000000000000            3976+         DC     4A(0)
001178 00000000                    3977+TEMPLATE DC     X'00000000'.            INITIALIZED FLAGS
00117C 0000000100000000            3978+         DC     F'1,0'
001184 0000000000000000            3979+         DC     5F'0,0'
0011AC FF000000000011B4            3980+         DC     X'FF00000000',AL3(JSP)
000054                             3981+TYPLEN   EQU    *-TYPPCB
```

```
          SAMPLE OPERATING SYSTEM I       VERSION 2.00
    LOC   OBJECT CODE     ADDR1 ADDR2  STMT   SOUPCE STATEMENT
                                       3983+**********************************************************************
                                       3984+*                                                                    *
                                       3985+*                          JOB STREAM PROCESSOR                      *
                                       3986+*                                                                    *
                                       3987+**********************************************************************

  0011B4                               3989+JSP      EQU   *.                          THE JOB STREAM PROCESSOR
  0011B4 0510                          3990+         BALR  1,0 .                        (PROCESS *IBSUP)
  0011B6                               3991+         USING *,1 .                       ESTABLISH ADDRESSING
  0011B6 4120 13AA            01560    3992+         LA    2,JSPSUSEM .                LOCK OURSELVES UNTIL
  0011BA 0AD7                          3993+         SVC   C'P' .                       WE CAN ALLOCATE STORAGE
  0011BC 4120 13B2            01568    3994+         LA    2,JSPAAS .                  READY TO ALLOCATE
  000000                               3995+         USING XAX,2
  0011C0 0AC5                          3996+         SVC   C'E' .                      ALLOCATE
  0011C2 58C0 2004            00004    3997+         L     12,XAXADDR .                PTR TO AUTO AREA
                                       3998+         DROP  2
  000000                               3999+         USING JSPAS,12 .                  USE FOR ADDRESSING
  0011C6 4120 13AA            01560    4000+         LA    2,JSPSUSEM .                UNLOCK OURSELVES
  0011CA 0AE5                          4001+         SVC   C'V'
  0011CC D207 C164 1422 00164 015D8    4002+         MVC   TREAD+0(8),=CL8'*IN' .      INITIALIZE VALUES IN AUTOMATIC
  0011D2 D203 C16C 144E 0016C 01604    4003+         MVC   TREAD+8(4),=F'8' .                      STORAGE
  0011D8 D203 C170 1452 00170 01608    4004+         MVC   TREAD+12(4),=C'READ'
  0011DE 4120 C084            00084    4005+         LA    2,CARD
  0011E2 5020 C174            00174    4006+         ST    2,ACARD
  0011E6 D207 C190 142A 00190 015E0    4007+         MVC   USERL+0(8),=CL8'USERPROG'
  0011EC D20B C178 1352 00178 01508    4008+         MVC   WRITE(12),SKIP
  0011F2 D203 C184 1456 00184 0160C    4009+         MVC   WRITE+12(4),=C'PRIN'
  0011F8 4150 C000            00000    4010+         LA    5,LINE
  0011FC 5050 C188            00188    4011+         ST    5,WRITE+16
  001200 D203 C1B0 144A 001B0 01600    4012+         MVC   CORE+8(4),=F'2048'
  001206 D207 C1B8 142A 001B8 015E0    4013+         MVC   TALK+0(8),=CL8'USERPROG'
  00120C D203 C1C0 145A 001C0 01610    4014+         MVC   TALK+8(4),=F'12'
  001212 D203 C1D8 145E 001D8 01614    4015+         MVC   ANYBACK+8(4),=F'1'
  001218 D203 C1B4 1462 001B4 01618    4016+         MVC   RLDTEMP,=A(0)
  00121E 5040 C18C            0018C    4017+         ST    4,KEY .                     STORE KEY
  001222 1853                          4018+         LR    5,3 .                       GET PTR TO UCB PTR BLOCK
  001224 5835 0000            00000    4019+         L     3,0(5) .                    GET READER POINTER
  001228 4120 1362            01518    4020+         LA    2,INSEQ .                   READY TO CREATE & START *IN
  00122C 0AC3                          4021+         SVC   C'C' .                      CREATE
  00122E 0AE8                          4022+         SVC   C'Y' .                      START
  001230 5835 0004            00004    4023+         L     3,4(5) .                    GET PTR TO PRINTER UCB
  001234 4120 136E            01524    4024+         LA    2,OUTSEQ .                  READY TO CREATE & START *OUT
  001238 0AC3                          4025+         SVC   C'C' .                      CREATE
  00123A 0AE8                          4026+         SVC   C'Y' .                      START

  00123C 4120 C164            00164    4028+LOOP     LA    2,TREAD .                   READY TO READ A CARD
  001240 0AE2                          4029+         SVC   C'S' .                      START TO READ
  001242 D203 C0DC 1466 000DC 0161C    4030+         MVC   RREPLY1,=F'132' .           132 CHARS FOR REPLY
  001248 4120 C0D4            000D4    4031+         LA    2,RREPLY
  00124C 0AD9                          4032+         SVC   C'R' .                      LISTEN FOR REPLY
  00124E D501 C0E0 1476 000E0 0162C    4033+         CLC   REPLY(2),=C'OK' .           IS REPLY 'OK'?
  001254 4770 10B0            01266    4034+         BNE   STOP .                      IF NOT, STOP
  001258 D504 1478 C084 0162E 00084    4035+         CLC   =C'$JOB,',CARD .            HAVE WE A JOB CARD?
  00125E 4780 10B6            0126C    4036+         BE    JOB .                       GOOD!
  001262 47F0 1086            0123C    4037+         B     LOOP .                      ELSE LOOP
```

```
         SAMPLE OPERATING SYSTEM      VERSION 2.00
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT
001266 4120 134A            01500  4038+STOP     LA    2,JSPNEVER .             WAIT FOR A "V" OPERATION
00126A 0AD7                        4039+         SVC   C'P' .                    THAT NEVER COMES

00126C 9200 C1DD      001DD        4041+JOB      MVI   LOADED,X'00' .           REMEMBER NOT LOADED
001270 D283 C000 1432 00000 015E8  4042+         MVC   LINE,=CL8' ' .           CLEAR A LINE, PUT IN
001276 D27B C008 C007 00008 00007  4043+         MVC   LINE+8(124),LINE+7 .     ALL BLANKS
00127C D24F C000 C084 00000 00084  4044+         MVC   LINE(80),CARD .          GET READY TO SEND $JOB CARD
001282 4120 C178            00178  4045+         LA    2,WRITE .                TO PRINTER
001286 0AE2                        4046+         SVC   C'S' .                   SEND IT
001288 4120 C0D4            000D4  4047+         LA    2,RREPLY
00128C 0AD9                        4048+         SVC   C'R' .                   AND WAIT FOR REPLY
00128E 4120 C190            00190  4049+         LA    2,USERL .                CREATE USERPROG
001292 0AC3                        4050+         SVC   C'C'
001294 4140 C088            00088  4051+         LA    4,CARD+4 .               START TO SCAN CARD
001298 4530 131A            014D0  4052+         BAL   3,SCAN .                 GET NEXT TOKEN
00129C 4180 0800            00800  4053+         LA    8,2048 .                 FIRST GUESS AT CORE SIZE
0012A0 5870 13A6            0155C  4054+         L     7,CORETABS .             GET CORE TABLE SIZE
0012A4 4160 137A            01530  4055+         LA    6,CORETAB .              INDEX INTO CORE TABLE
0012A8 4450 110A            012C0  4056+CORELOOP EX    5,CORECOMP .             IS THIS THE CORE SPEC?
0012AC 4780 1110            012C6  4057+         BE    COREOK .                 IF SO, BINGO!
0012B0 4188 0800            00800  4058+         LA    8,2048(8) .              ELSE UP OUR GUESS
0012B4 4166 0004            00004  4059+         LA    6,4(6) .                  AND OUR INDEX
0012B8 4670 10F2            012A8  4060+         BCT   7,CORELOOP .              AND TRY AGAIN
0012BC 47F0 12CA            01480  4061+         B     EXPUNGE .                ELSE THROW HIM AWAY
0012C0 D500 9000 6000 00000 00000  4062+CORECOMP CLC   0(0,9),0(6) .            EX'D TO TEST AGAINST CORE TABLE
0012C6 5080 C1A8            001A8  4063+COREOK   ST    8,CORE .                 REMEMBER CORE REQUIREMENT
0012CA 4530 131A            014D0  4064+ASGNUNIT BAL   3,SCAN .                 GET NEXT TOKEN
0012CE 957E 4000      00000        4065+         CLI   0(4),C'=' .              IS IT AN '='?
0012D2 4770 119C            01352  4066+         BNE   LOAD .                   IF NOT, LOAD IN THE OBJECT DECK
0012D6 955C 9000      00000        4067+         CLI   0(9),C'*' .              HAS USER NAMED IT STARTING
0012DA 4780 12CA            01480  4068+         BE    EXPUNGE .                 WITH * ? IF SO, THROW HIM OUT
0012DE 4120 C19C            0019C  4069+         LA    2,SEQ .                  ELSE CREATE A PROCESS
0012E2 D207 C19C 1432 0019C 015E8  4070+         MVC   SEQ,=CL8' ' .            BLANK OUT THE NAME
0012E8 4450 115C            01312  4071+         EX    5,UNAMMOV .              THEN MOVE THE RELEVANT
0012EC 0AC3                        4072+         SVC   C'C' .                    CHARACTERS AND CREATE
0012EE 4120 C19C            0019C  4073+         LA    2,SEQ .                  WE'LL START IT IN A MOMENT
0012F2 4530 131A            014D0  4074+         BAL   3,SCAN .                 SCAN AGAIN
0012F6 4450 1162            01318  4075+         EX    5,CMPIN .                IS IT 'IN'?
0012FA 4780 1174            0132A  4076+         BE    ASIN .                   IF SO, ASSIGN IT AS IN
0012FE 4450 1168            0131E  4077+         EX    5,CMPOUT .               IF IT'S 'OUT',
001302 4780 1184            0133A  4078+         BE    ASOUT .                   ASSIGN IT AS OUT
001306 4450 116E            01324  4079+         EX    5,CMPEXCP .              IS IT 'EXCP'?
00130A 4780 118C            01342  4080+         BE    ASEXCP .                  IF SO, ASSIGN IT AS EXCP
00130E 47F0 12CA            01480  4081+         B     EXPUNGE .                ERROR: GO ON TO NEXT JOB
001312 D200 C19C 9000 0019C 00000  4082+UNAMMOV  MVC   SEQ(0),0(9) .            MOVE THE UNIT'S PROCESS NAME
001318 D500 9000 147D 00000 01633  4083+CMPIN    CLC   0(0,9),=C'IN ' .         DOES IT SAY 'IN'?
00131E D500 9000 146A 00000 01620  4084+CMPOUT   CLC   0(0,9),=C'OUT ' .        DOES IT SAY 'OUT'?
001324 D500 9000 1480 00000 01636  4085+CMPEXCP  CLC   0(0,9),=C'EXCP ' .       DOES IT SAY 'EXCP'?

00132A 41B0 1422            015D8  4087+ASIN     LA    11,=CL8'*IN' .           POINT TO NAME OF READER HANDLER
00132E D203 C1A4 146E 001A4 01624  4088+SETDIM   MVC   UNITRTN,=A(DIM) .        USE A DIM AS THE INTERFACE
001334 0AE8                        4089+         SVC   C'Y'
001336 47F0 1114            012CA  4090+         B     ASGNUNIT
00133A 41B0 143A            015F0  4091+ASOUT    LA    11,=CL8'*OUT' .          POINT TO NAME OF PRINTER HANDLER
00133E 47F0 1178            0132E  4092+         B     SETDIM
```

```
             SAMPLE OPERATING SYSTEM         VERSION 2.00
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT    SOURCE STATEMENT
001342 D203 C1A4 1472 001A4 01628  4093+ASEXCP   MVC   UNITRTN,=A(EXCPHNDL) .  USE FOR USER SUPPLIED
001348 58B0 C18C            0018C  4094+         L     11,KEY
00134C 0AE8                        4095+         SVC   C'Y' .                  I/O ROUTINE
00134E 47F0 1114            012CA  4096+         B.    ASGNUNIT

001352 4120 C1A8            001A8  4098+LOAD     LA    2,CORE .                READY TO ALLOCATE THE REGION
001356 0AC1                        4099+         SVC   C'A' .                  AND ALLOCATE IT
001358 92FF C1DD      001DD        4100+         MVI   LOADED,X'FF' .          REMEMBER THAT WE'RE LOADED
00135C 5890 C1AC            001AC  4101+         L     9,CORE+4 .              GET THE FIRST ADDRESS
001360 5840 C18C            0018C  4102+         L     4,KEY .                 GET THE KEY
001364 8840 0010            00010  4103+         SRL   4,16
001368 1839                        4104+         LR    3,9 .                   GET THE BLOCK FOLLOWING OURS
00136A 1A38                        4105+         AR    3,8
00136C 5B30 144A            01600  4106+LOADSK   S     3,=F'2048' .            GET THE PREVIOUS BLOCK
001370 1939                        4107+         CR    3,9 .                   HAVE WE PASSED THE START?
001372 4740 11C6            0137C  4108+         BL    LOADLOOP .              IF SO, START LOADING
001376 0843                        4109+         SSK   4,3 .                   ELSE SET THIS BLOCK TO THE KEY,
001378 47F0 11B6            0136C  4110+         B     LOADSK .                AND BRANCH BACK
00137C 4120 C164            00164  4111+LOADLOOP LA    2,TREAD .               READ IN OBJECT DECK.
001380 0AE2                        4112+         SVC   C'S' .                  GET A CARD A'READING.
001382 D203 C0DC 1466 000DC 0161C  4113+         MVC   RREPLY1,=F'132'
001388 4120 C0D4            000D4  4114+         LA    2,RREPLY
00138C 0AD9                        4115+         SVC   C'R' .                  WAIT FOR ANSWER
00138E D502 C085 1485 00085 0163B  4116+         CLC   CARD+1(3),=C'TXT' .     IS IT A TXT CARD?
001394 4780 11FA            013B0  4117+         BE    TXTCARD
001398 D502 C085 1488 00085 0163E  4118+         CLC   CARD+1(3),=C'RLD' .     IS IT A RLD CARD?
00139E 4780 1214            013CA  4119+         BE    RLDCARD
0013A2 D502 C085 148B 00085 01641  4120+         CLC   CARD+1(3),=C'END' .     IS IT AN END CARD?
0013A8 4780 1272            01428  4121+         BE    ENDCARD
0013AC 47F0 11C6            0137C  4122+         B     LOADLOOP .              IF NONE, IGNORE.

0013B0 58A0 C088            00088  4124+TXTCARD  L     10,CARD+4 .             GET THE RELATIVE ADDRESS
0013B4 1AA9                        4125+         AR    10,9 .                  PLUS THE ABSOLUTE ADDRESS
0013B6 48B0 C08E            0008E  4126+         LH    11,CARD+10 .            GET THE COUNT,
0013BA 06B0                        4127+         BCTR  11,0 .                   DECREMENTED.
0013BC 44B0 120E            013C4  4128+         EX    11,TXTMOV .             AND MOVE THE TEXT.
0013C0 47F0 11C6            0137C  4129+         B     LOADLOOP .              AND READ ANOTHER CARD! OH WOW!
0013C4 D200 A000 C094 00000 00094  4130+TXTMOV   MVC   0(0,10),CARD+16

0013CA 48B0 C08E            0008E  4132+RLDCARD  LH    11,CARD+10 .            GET THE BYTE COUNT
0013CE 41D0 C098            00098  4133+         LA    13,CARD+20 .            AND AN INDEX INTO THE CARD
0013D2 58AD 0000            00000  4134+RLDLOOP  L     10,0(13) .              GET THE LOCATION TO BE RLD'D
0013D6 1AA9                        4135+         AR    10,9 .                  GET THE ABSOLUTE ADDRESS
0013D8 9103 D003      00003        4136+         TM    3(13),X'03' .           IS IT A FULLWORD?
0013DC 4770 1254            0140A  4137+         BNZ   NOTALGND .              IF NO, HANDLE AS THREE BYTES
0013E0 587A 0000            00000  4138+         L     7,0(10) .               GET THAT WORD (HAD BETTER BE
0013E4 1A79                        4139+         AR    7,9 .                    ONE); ADD THE RELOCATION
0013E6 507A 0000            00000  4140+         ST    7,0(10) .                ADDRESS, AND STORE IT BACK
0013EA 9101 D000      00000        4141+RLDCONT  TM    0(13),X'01' .           CHECK IF LONG OR SHORT FIELD
0013EE 4770 1244            013FA  4142+         BNZ   SHORT .                 AND BRANCH ACCORDINGLY
0013F2 4140 0008            00008  4143+         LA    4,8 .                   SKIP EIGHT BYTES
0013F6 47F0 1248            013FE  4144+         B     RLDFINI
0013FA 4140 0004            00004  4145+SHORT    LA    4,4 .                   SKIP FOUR BYTES
0013FE 1AD4                        4146+RLDFINI  AR    13,4 .                  INCREMENT THE CARD INDEX
001400 1BB4                        4147+         SR    11,4 .                  DECREMENT THE BYTE COUNT
```

```
          SAMPLE OPERATING SYSTEM!    VERSION 2.00
LOC    OBJECT CODE    ADDR1 ADDR2  STMT    SOURCE STATEMENT
001402 4720 121C            013D2  4148+            BP    RLDLOOP .                     AND TRY AGAIN
001406 47F0 11C6            0137C  4149+            B     LOADLOOP .                    OR READ ANOTHER CARD
00140A D202 C1B5 A000 001B5 00000  4150+NOTALGND MVC   RLDTEMP+1(3),0(10) .          PUT ADDRESS HERE
001410 5870 C1B4            001B4  4151+            L     7,RLDTEMP .                   RELOCATE IT
001414 1A79                        4152+            AR    7,9
001416 5070 C1B4            001B4  4153+            ST    7,RLDTEMP .                   AND PUT IT BACK TO
00141A D202 A000 C1B5 00000 001B5  4154+            MVC   0(3,10),RLDTEMP+1 .            WHERE IT BELONGS
001420 9400 C1B4      001B4        4155+            NI    RLDTEMP,X'00' .               CLEAR OUT TEMPORARY
001424 47F0 1234            013EA  4156+            B     RLDCONT .                     AND LOOP BACK

001428 4120 C190            00190  4158+ENDCARD     LA    2,USERL .                     FIND THE PCB FOR USERPROG
00142C 0AD5                        4159+            SVC   C'N'
00142E 5840 C198            00198  4160+            L     4,USERL+8 .                   GET THE ADDRESS,
000000                             4161+            USING PCB,4
001432 92FF 4019      00019        4162+            MVI   PCBBLOKT,X'FF' .              TEMPORARILY BLOCK IT
001436 5090 C198            00198  4163+            ST    9,USERL+8 .                   STORE THE BEGINNING ADDRESS
00143A 0AE8                        4164+            SVC   C'Y' .                        THEN START IT
00143C 5850 C18C            0018C  4165+            L     5,KEY .                       GET THE KEY,
001440 5650 404C            0004C  4166+            O     5,PCBISA+0 .                  THEN OR THIS INTO THE
001444 5050 404C            0004C  4167+            ST    5,PCBISA+0 .                   FIRST WORD OF THE PCB
001448 9601 404D      0004D        4168+            OI    PCBISA+1,X'01' .              OR IN A 'PROGRAM STATE' BIT
00144C 9200 4019      00019        4169+            MVI   PCBBLOKT,X'00' .              AND THEN UNBLOCK IT
                                   4170+            DROP  4
001450 4120 C1B8            001B8  4171+            LA    2,TALK .                      LISTEN TO WHAT IT SAYS
001454 0AD9                        4172+            SVC   C'R'

001456 D207 C000 1432 00000 015E8  4174+            MVC   LINE(8),=CL8' ' .             JOB FINISHED, CLEAR A LINE
00145C D27B C008 C007 00008 00007  4175+            MVC   LINE+8(124),LINE+7
001462 D20B C000 C1C4 00000 001C4  4176+            MVC   LINE(12),TALK+12 .            MOVE THE MESSAGE ONTO THE LINE
001468 4120 C178            00178  4177+            LA    2,WRITE .                     AND SAY TO WRITE IT
00146C 0AE2                        4178+            SVC   C'S'
00146E 4120 C1D0            001D0  4179+            LA    2,ANYBACK .                   WAIT FOR THE ANSWER
001472 0AD9                        4180+            SVC   C'R'
001474 4120 1352            01508  4181+            LA    2,SKIP .                      SKIP TO THE TOP OF THE NEXT PAGE
001478 0AE2                        4182+            SVC   C'S'
00147A 4120 C1D0            001D0  4183+            LA    2,ANYBACK
00147E 0AD9                        4184+            SVC   C'R'

001480 5850 0270            00270  4186+EXPUNGE     L     5,RUNNING .                   EXPUNGE A JOB: LOOK AT ALL PCBS
001484 4120 C19C            0019C  4187+            LA    2,SEQ
000000                             4188+            USING PCB,5
001488 D207 C19C 5000 0019C 00000  4189+EXPLOOP     MVC   SEQ(8),PCBNAME .              GET THE PROCESS NAME
00148E 5840 5008            00008  4190+            L     4,PCBNPTG .                   GET THE NEXT PTR
001492 955C C19C      0019C        4191+            CLI   SEQ+0,C'*' .                  IS IT A '*' PROCESS?
001496 4780 12E8            0149E  4192+            BE    EXPNXT .                      IF SO, SKIP OVER
00149A 0AE9                        4193+            SVC   C'Z' .                        ELSE STOP IT
00149C 0AC4                        4194+            SVC   C'D' .                        AND DESTROY IT
00149E 1854                        4195+EXPNXT      LR    5,4 .                         GO TO THE NEXT PCB
0014A0 5950 0270            00270  4196+            C     5,RUNNING .                   ARE WE THROUGH?
0014A4 4770 12D2            01488  4197+            BNE   EXPLOOP .                     IF NOT, LOOP AGAIN
0014A8 9500 C1DD      001DD        4198+            CLI   LOADED,X'00' .                WAS CORE ALLOCATED?
0014AC 4780 1086            0123C  4199+            BE    LOOP .                        IF NOT, GO READ NEXT $JOB CARD
0014B0 1B44                        4200+            SR    4,4 .                         ELSE GET A ZERO KEY
0014B2 1839                        4201+            LR    3,9 .                         AND A POINTER TO THE NEXT
0014B4 1A38                        4202+            AR    3,8 .                          BLOCK AFTER OURS.
```

```
            SAMPLE OPERATING SYSTEM      VERSION 2.00

   LOC    OBJECT CODE      ADDR1 ADDR2  STMT    SOURCE STATEMENT

  0014B6 5B30 144A               01600  4203+LOADCL    S      3,=F'2048' .                  GO TO THE PREVIOUS BLOCK
  0014BA 1939                           4204+          CR     3,9 .                         ARE WE THROUGH?
  0014BC 4740 1310               014C6  4205+          BL     LOADD .                       IF SO, GO FREE CORE
  0014C0 0843                           4206+          SSK    4,3 .                         ELSE CLEAR STORAGE KEY
  0014C2 47F0 1300               014B6  4207+          B      LOADCL .                      AND LOOP BACK
  0014C6 4120 C1A8               001A8  4208+LOADD     LA     2,CORE
  0014CA 0AC6                           4209+          SVC    C'F' .                        FREE THE STORAGE
  0014CC 47F0 1086               0123C  4210+          B      LOOP .                        READ ANOTHER $JOB CARD

  0014D0 1B55                           4212+SCAN      SR     5,5 .                         START TOKEN COUNT AT ZERO
  0014D2 4144 0001               00001  4213+SCANLOOP  LA     4,1(4) .                      GO TO NEXT CHARACTER
  0014D6 956B 4000         00000        4214+          CLI    0(4),C',' .                   DO WE HAVE A DELIMITER? OF SO,
  0014DA 4780 1340               014F6  4215+          BE     TOKSTART
  0014DE 957E 4000         00000        4216+          CLI    0(4),C'=' .                   DITTO
  0014E2 4780 1340               014F6  4217+          BE     TOKSTART
  0014E6 9540 4000         00000        4218+          CLI    0(4),C' ' .                   DITTO
  0014EA 4780 1340               014F6  4219+          BE     TOKSTART
  0014EE 4155 0001               00001  4220+          LA     5,1(5) .                      AND UP COUNT
  0014F2 47F0 131C               014D2  4221+          B      SCANLOOP .                    AND LOOP
  0014F6 1894                           4222+TOKSTART  LR     9,4 .                         SET REG9 TO START
  0014F8 1B95                           4223+          SR     9,5 .                         OF THIS TOKEN
  0014FA 0650                           4224+          BCTR   5,0 .                         LESS ONE FOR EXECUTE INSTRUCTION
  0014FC 07F3                           4225+          BR     3

  0014FE 0000
  001500 0000000000000000               4227+JSPNEVER  DC     F'0,0' .                      A GOOD WAY TO DIE: P(JSPNEVER)
  001508 5CD6E4E340404040               4228+SKIP      DC     CL8'*OUT' .                   MESSAGE BLOCK FOR NEW PAGE
  001510 00000008                       4229+          DC     F'8'
  001514 E2E3C3F1                       4230+          DC     CL4'STC1'
  001518 5CC9D54040404040               4231+INSEQ     DC     CL8'*IN' .                    SEQ TO CREATE & START *IN
  001520 00000AC6                       4232+          DC     A(RDRHANDL)
  001524 5CD6E4E340404040               4233+OUTSEQ    DC     CL8'*OUT' .                   SEQ TO CREATE & START *OUT
  00152C 00000C24                       4234+          DC     A(PRTHANDL)
  001530                                4235+CORETAB   EQU    * .                           TABLE OF CORE SIZES
  001530 F2D24040                       4236+          DC     CL4'2K'
  001534 F4D24040                       4237+          DC     CL4'4K'
  001538 F6D24040                       4238+          DC     CL4'6K'
  00153C F8D24040                       4239+          DC     CL4'8K'
  001540 F1F0D240                       4240+          DC     CL4'10K'
  001544 F1F2D240                       4241+          DC     CL4'12K'
  001548 F1F4D240                       4242+          DC     CL4'14K'
  00154C F1F6D240                       4243+          DC     CL4'16K'
  001550 F1F8D240                       4244+          DC     CL4'18K'
  001554 F2F0D240                       4245+          DC     CL4'20K'
  001558 F2F2D240                       4246+          DC     CL4'22K'
  00155C 0000000B                       4247+CORETABS  DC     A((*-CORETAB)/4) .            CORE TABLE SIZE
  001560 0000000100000000               4248+JSPSUSEM  DC     F'1,0' .                      SEMAPHORE TO LOCK ROUTINE
  001568 000001E0                       4249+JSPAAS    DC     A(LENJSPAS) .                 ALLOCATE LIST FOR AUTO STORAGE
  00156C                                4250+          DS     A
  001570 00000008                       4251+          DC     F'8'
```

```
          SAMPLE OPERATING SYSTEM      VERSION 2.00

   LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT
                                     4253+**********************************************************************
                                     4254+*                                                                    *
                                     4255+*                       DEVICE INTERFACE MODULE                      *
                                     4256+*                                                                    *
                                     4257+*       FUNCTION: TO INTERFACE BETWEEN USERPROG AND DEVICE HANDLER   *
                                     4258+*      DATABASES: NONE                                               *
                                     4259+*  ROUTINES USED: XA, XP, XV, XR, XS                                 *
                                     4260+*      PROCEDURE: ALLOCATE AUTOMATIC STORAGE; START TO READ MESSAGE  *
                                     4261+*                 FROM USER; SEND MESSAGE TO DEVICE HANDLER;         *
                                     4262+*                 CONTINUE LOOPING, SENDING MESSAGES FROM USER TO    *
                                     4263+*                 DEVICE HANDLER AND BACK.                           *
                                     4264+*   ERROR CHECKS: NONE                                               *
                                     4265+*     INTERRUPTS: ON                                                 *
                                     4266+*    USER ACCESS: YES                                                *
                                     4267+*                                                                    *
                                     4268+**********************************************************************
001574                               4270+DIM      EQU   * .                    THE DEVICE INTERFACE MODULE
001574 0510                          4271+         BALR  1,0
001576                               4272+         USING *,1 .                  ESTABLISH ADDRESSING
001576 4120 1042              015B8  4273+         LA    2,DIMSEM .             LOCK UNTIL GET STORAGE
00157A 0AD7                          4274+         SVC   C'P'
00157C 4120 104A              015C0  4275+         LA    2,DIMAAS .             READY TO ALLOCATE STORAGE
000000                               4276+         USING XAX,2
001580 0AC5                          4277+         SVC   C'E' .                 DO IT
001582 58C0 2004              00004  4278+         L     12,XAXADDR .           GET THE ADDRESS
                                     4279+         DROP  2
001586 4120 1042              015B8  4280+         LA    2,DIMSEM .             UNLOCK OURSELVES
00158A 0AE5                          4281+         SVC   C'V'
000000                               4282+         USING DIMAS,12 .             USE 12 FOR AUTO STORAGE
00158C D207 C090 B000 00090 00000    4283+         MVC   DIMLMS,C(11) .         MOVE NAME OF RECEIVER
001592 4180 0084              00084  4284+         LA    8,132 .                REG 8 = SIZE OF MESSAGE
001596 5080 C008              00008  4285+DIMLOOP  ST    8,DIMMSG+8 .           GET READY TO READ A MESSAGE
00159A 4120 C000              00000  4286+         LA    2,DIMMSG
00159E 0AD9                          4287+         SVC   C'R' .                 READ
0015A0 D207 C098 C000 00098 00000    4288+         MVC   DIMTEMP,DIMMSG .       SAVE SENDER NAME
0015A6 D207 C000 C090 00000 00090    4289+         MVC   DIMMSG,DIMLMS .        SEND IT BACK TO THE LAST GUY
0015AC 0AE2                          4290+         SVC   C'S' .                 SEND IT
0015AE D207 C090 C098 00090 00098    4291+         MVC   DIMLMS,DIMTEMP .       AND REMEMBER WHO TO SEND TO NEXT
0015B4 47F0 1020              01596  4292+         B     DIMLOOP .              RELOOP
0015B8 0000000100000000              4293+DIMSEM   DC    F'1,0' .               SEMAPHORE FOR ENTRY
0015C0 000000A0                      4294+DIMAAS   DC    A(DIMLEN) .            ALLOCATE SEQ FOR AUTO STORAGE
0015C4 00000000                      4295+         DC    A(0)
0015C8 00000008                      4296+         DC    F'8'
                                     4297+         DROP  12
```

```
          SAMPLE OPERATING SYSTEM    VERSION 2.00

  LOC  OBJECT CODE    ADDR1 ADDR2 STMT   SOURCE STATEMENT

0015D0                            4299+         LTORG
0015D0 00000000000069B8           4300                 =A(0,CORESIZE-(VERYEND-PROGRAM))
0015D8 5CC9D54040404040           4301                 =CL8'*IN'
0015E0 E4E2C5D9D7D9D6C7           4302                 =CL8'USERPROG'
0015E8 4040404040404040           4303                 =CL8' '
0015F0 5CD6E4E340404040           4304                 =CL8'*OUT'
0015F8 00000E74                   4305                 =A(UCBTABLE)
0015FC 00000EF4                   4306                 =A(UCBTBEND)
001600 00000800                   4307                 =F'2048'
001604 00000008                   4308                 =F'8'
001608 D9C5C1C4                   4309                 =C'READ'
00160C D7D9C9D5                   4310                 =C'PRIN'
001610 0000000C                   4311                 =F'12'
001614 00000001                   4312                 =F'1'
001618 00000000                   4313                 =A(0)
00161C 00000084                   4314                 =F'132'
001620 D6E4E340                   4315                 =C'OUT '
001624 00001574                   4316                 =A(DIM)
001628 00000D44                   4317                 =A(EXCPHNDL)
00162C D6D2                       4318                 =C'OK'
00162E 5BD1D6C26B                 4319                 =C'$JOB,'
001633 C9D540                     4320                 =C'IN '
001636 C5E7C3D740                 4321                 =C'EXCP '
00163B E3E7E3                     4322                 =C'TXT'
00163E D9D3C4                     4323                 =C'RLD'
001641 C5D5C4                     4324                 =C'END'
001648                            4325+VERYEND  DS     D .                 BEGINNING OF FREE STORAGE
```

```
           SAMPLE OPERATING SYSTEM     VERSION 2.00

     LOC  OBJECT CODE    ADDR1 ADDR2 STMT   SOURCE STATEMENT
                                     4327+*****************************************************************
                                     4328+*                                                               *
                                     4329+*                          DATABASE DEFINITIONS                 *
                                     4330+*                                                               *
                                     4331+*****************************************************************

   000000                            4333+PCB      DSECT  .                      PROCESS CONTROL BLOCK DEFINITION
   000000                            4334+PCBNAME  DS     CL8 .                  NAME
   000008                            4335+PCBNPTG  DS     F .                    NEXT POINTER THIS GROUP
   00000C                            4336+PCBLPTG  DS     F .                    LAST POINTER THIS GROUP
   000010                            4337+PCBNPALL DS     F .                    NEXT POINTER ALL
   000014                            4338+PCBLPALL DS     F .                    LAST POINTER ALL
   000018                            4339+PCBSTOPT DS     C .                    STOPPED
   000019                            4340+PCBBLOKT DS     C .                    BLOCKED
   00001A                            4341+PCBINSMC DS     C .                    IN SMC
   00001B                            4342+PCBSW    DS     C .                    STOP WAITING
   00001C                            4343+PCBMSC   DS     CL8 .                  MESSAGE SEMAPHORE COMMON
   000024                            4344+PCBMSR   DS     CL8 .                  MESSAGE SEMAPHORE RECEIVER
   00002C                            4345+PCBFM    DS     F .                    FIRST MESSAGE
   000030                            4346+PCBNSW   DS     F .                    NEXT SEMAPHORE WAITER
   000034                            4347+PCBSRS   DS     CL8 .                  STOPPER SEMAPHORE
   00003C                            4348+PCBSES   DS     CL8 .                  STOPPEE SEMAPHORE
   000044                            4349+PCBASIZE DS     F .                    AUTOMATIC STORAGE SIZE
   000048                            4350+PCBAADDR DS     A .                    AUTOMATIC STORAGE ADDRESS
   00004C                            4351+PCBISA   DS     CL84 .                 INTERRUPT SAVE AREA
   0000A0                            4352+PCBFSA   DS     CL84 .                 FAULT SAVE AREA
   0000F4                            4353+PCBMSA   DS     CL84 .                 MEMORY SAVE AREA
   000148                            4354+         DS     0D .                   (ALIGN)
   000148                            4355+LENPCB   EQU    *-PCB .                (LENGTH)

   000000                            4357+SA       DSECT  .                      SAVE AREA DEFINITION
   000000                            4358+SAPSW    DS     D .                    PROGRAM STATUS WORD
   000008                            4359+SAREGS   DS     CL64 .                 REGISTERS
   000048                            4360+SATEMP   DS     CL12 .                 TEMPORARIES

   000000                            4362+REGS     DSECT  .                      REGISTERS DEFINITION
   000000                            4363+REG0     DS     F .                    REGISTER 0
   000004                            4364+REG1     DS     F .                    REGISTER 1
   000008                            4365+REG2     DS     F .                    REGISTER 2
   00000C                            4366+REG3     DS     F .                    REGISTER 3
   000010                            4367+REG4     DS     F .                    REGISTER 4
   000014                            4368+REG5     DS     F .                    REGISTER 5
   000018                            4369+REG6     DS     F .                    REGISTER 6
   00001C                            4370+REG7     DS     F .                    REGISTER 7
   000020                            4371+REG8     DS     F .                    REGISTER 8
   000024                            4372+REG9     DS     F .                    REGISTER 9
   000028                            4373+REG10    DS     F .                    REGISTER 10
   00002C                            4374+REG11    DS     F .                    REGISTER 11
   000030                            4375+REG12    DS     F .                    REGISTER 12
   000034                            4376+REG13    DS     F .                    REGISTER 13
   000038                            4377+REG14    DS     F .                    REGISTER 14
   00003C                            4378+REG15    DS     F .                    REGISTER 15
```

```
                  SAMPLE OPERATING SYSTEM    VERSION 2.00

   LOC  OBJECT CODE    ADDR1 ADDR2  STMT    SOURCE STATEMENT

 000000                             4380+FSB        DSECT  .                      FREE STORAGE BLOCK DEFINITIONS
 000000                             4381+FSBNEXT    DS     A .                    NEXT
 000004                             4382+FSBSIZE    DS     F .                    SIZE

 000000                             4384+SM         DSECT  .                      SEMAPHORE DEFINITION
 000000                             4385+SMVAL      DS     F .                    VALUE
 000004                             4386+SMPTR      DS     F .                    PTR

 000000                             4388+MSG        DSECT  .                      MESSAGE DEFINITION
 000000                             4389+MSGSENDR   DS     A .                    POINTER TO SENDER'S PCB
 000004                             4390+MSGNEXT    DS     A .                    NEXT
 000008                             4391+MSGSIZE    DS     F .                    SIZE
 00000C                             4392+MSGTEXT    DS     0C .                   TEXT
 00000C                             4393+LENMSG     EQU    *-MSG .                (LENGTH)

 000000                             4395+XAX        DSECT  .                      XA ARGUMENT LIST
 000000                             4396+XAXSIZE    DS     F .                    SIZE
 000004                             4397+XAXADDR    DS     F .                    ADDRESS
 000008                             4398+XAXALGN    DS     F .                    ALIGNMENT

 000000                             4400+XFX        DSECT  .                      XF ARGUMENT LIST
 000000                             4401+XFXSIZE    DS     F .                    SIZE
 000004                             4402+XFXADDR    DS     F .                    ADDRESS

 000000                             4404+XBX        DSECT  .                      XB ARGUMENT LIST
 000000                             4405+XBXSIZE    DS     F .                    SIZE
 000004                             4406+XBXADDR    DS     F .                    ADDRESS

 000000                             4408+XCX        DSECT  .                      XC ARGUMENT LIST
 000000                             4409+XCXNAME    DS     CL8 .                  NAME

 000000                             4411+XDX        DSECT  .                      XD ARGUMENT LIST
 000000                             4412+XDXNAME    DS     CL8 .                  NAME

 000000                             4414+XNX        DSECT  .                      XN ARGUMENT LIST
 000000                             4415+XNXNAME    DS     CL8 .                  NAME
 000008                             4416+XNXADDR    DS     A .                    ADDRESS

 000000                             4418+XRX        DSECT  .                      XR ARGUMENT LIST
 000000                             4419+XRXNAME    DS     CL8 .                  NAME
 000008                             4420+XRXSIZE    DS     F .                    SIZE
 00000C                             4421+XRXTEXT    DS     0C .                   TEXT

 000000                             4423+XSX        DSECT  .                      XS ARGUMENT LIST
 000000                             4424+XSXNAME    DS     CL8 .                  NAME
 000008                             4425+XSXSIZE    DS     F .                    SIZE
 00000C                             4426+XSXTEXT    DS     0C .                   TEXT

 000000                             4428+XYX        DSECT  .                      XY ARGUMENT LIST
 000000                             4429+XYXNAME    DS     CL8 .                  NAME
 000008                             4430+XYXADDR    DS     A .                    ADDR

 000000                             4432+XZX        DSECT  .                      XZ ARGUMENT LIST
 000000                             4433+XZXNAME    DS     CL8 .                  NAME
```

```
          SAMPLE OPERATING SYSTEM     VERSION 2.00
   LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

  000000                            4435+RDRHAS    DSECT   .                      READER HANDLER AUTOMATIC STORAGE
  000000                            4436+RDRHCCB   DS      2F .                   CCB
  000008                            4437+RDRHMSG   DS      CL8 .                  MESSAGE BLOCK FOR REQUESTS
  000010                            4438+          DS      F'8'
  000014                            4439+          DS      CL8
  00001C                            4440+RDRHTEMP  DS      CL80 .                 AREA FOR $JOB IN DATA STREAM
  00006C                            4441+RDRHM     DS      CL8 .                  MESSAGE BLOCK FOR REPLY
  000074                            4442+          DS      F'2'
  000078                            4443+          DS      CL2
  00007A                            4444+JOBBIT    DS      1C
  000080                            4445+          DS      0D
  000080                            4446+LENRDRHA  EQU     *-RDRHAS .             (LENGTH)

  000000                            4448+PRTHAS    DSECT   .                      PRINTER HANDLER AUTO STORAGE
  000000                            4449+PRTHCCB   DS      2F .                   CCB
  000008                            4450+PRTHMSG   DS      CL8 .                  MESSAGE BLOCK FOR REQUESTS
  000010                            4451+          DS      F'2'
  000014                            4452+          DS      CL8
  00001C                            4453+PRTHM     DS      CL8 .                  MESSAGE BLOCK FOR REPLYS
  000024                            4454+          DS      F'2'
  000028                            4455+          DS      CL2
  000030                            4456+          DS      0D
  000030                            4457+LENPRTHA  EQU     *-PRTHAS .             (LENGTH)

  000000                            4459+EXCPHAS   DSECT   .                      EXCP HANDLER AUTOMATIC STORAGE
  000000                            4460+EXCPHMSG  DS      CL8 .                  MESSAGE BLOCK FOR REQUESTS
  000008                            4461+          DS      F'12'
  00000C                            4462+          DS      CL12
  000018                            4463+EXCPHM    DS      CL8 .                  MESSAGE BLOCK FOR REPLY
  000020                            4464+          DS      F'12'
  000024                            4465+          DS      CL12
  000030                            4466+          DS      0D
  000030                            4467+LENEXCPA  EQU     *-EXCPHAS      .

  000000                            4469+UCB       DSECT   .                      UNIT CONTROL BLOCK DEFINITION
  000000                            4470+UCBADDR   DS      F .                    ADDRESS
  000004                            4471+UCBUS     DS      FL8 .                  USER SEMAPHORE
  00000C                            4472+UCBWS     DS      FL8 .                  WAITER SEMAPHORE
  000014                            4473+UCBCSW    DS      FL8 .                  CHANNEL STATUS WORD
  00001C                            4474+UCBFPR    DS      CL1 .                  FAST PROCCESSING REQUIRED
  000020                            4475+          DS      0F
  000020                            4476+UCBLENG   EQU     *-UCB

  000000                            4478+JSPAS     DSECT   .                      JSP AUTOMATIC STORAGE
  000000                            4479+LINE      DS      CL132 .                PRINTED LINE
  000084                            4480+          DS      0F
  000084                            4481+CARD      DS      CL80 .                 CARD READ
  0000D4                            4482+          DS      0F
  0000D4                            4483+RREPLY    DS      CL8 .                  MESSAGE BLOCK FOR REPLIES
  0000DC                            4484+RREPLY1   DS      F
  0000E0                            4485+REPLY     DS      CL132
  000164                            4486+TREAD     DS      0F .                   MESSAGE BLOCK FOR READING
  000164                            4487+          DS      CL8'*IN'
  00016C                            4488+          DS      F'8'
  000170                            4489+          DS      CL4'READ'
```

```
          SAMPLE OPERATING SYSTEM    VERSION 2.00

   LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

 000174                             4490+ACARD    DS     A(0)
 000178                             4491+WRITE    DS     CL8'*OUT' .             MESSAGE BLOCK TO PRINT A LINE
 000180                             4492+         DS     F'8'
 000184                             4493+         DS     CL4'PRIN'
 000188                             4494+         DS     A (LINE)
 00018C                             4495+KEY      DS     F
 000190                             4496+USERL    DS     CL8'USERPROG' .         LIST FOR MANIPULATING USERPROG
 000198                             4497+         DS     F
 00019C                             4498+SEQ      DS     CL8' ' .                COMMON ARG LIST FOR I/O PROCESS
 0001A4                             4499+UNITRTN  DS     A
 0001A8                             4500+CORE     DS     F .                     MEMORY ALLOCATE AND FREE
 0001AC                             4501+         DS     F .                     SEQUENCE
 0001B0                             4502+         DS     F'2048'
 0001B4                             4503+RLDTEMP  DS     F
 0001B8                             4504+TALK     DS     CL8'USERPROG' .         MESSAGE BLOCK FOR MESSAGE FROM
 0001C0                             4505+         DS     F'12' .                  USERPROG
 0001C4                             4506+         DS     CL12
 0001D0                             4507+ANYBACK  DS     CL8 .                   MESSAGE BLOCK FOR IGNORING MESS
 0001D8                             4508+         DS     F'1'
 0001DC                             4509+         DS     CL1
 0001DD                             4510+LOADED   DS     C .                     IF CORE ALLOCATED
 0001E0                             4511+         DS     0D
 0001E0                             4512+LENJSPAS EQU    *-JSPAS .               (LENGTH)

 000000                             4514+DIMAS    DSECT  .                       DEVICE INTERFACE MODULE STORAGE
 000000                             4515+DIMMSG   DS     CL8 .                   MESSAGE BLOCK
 000008                             4516+         DS     F'132'
 00000C                             4517+         DS     CL132
 000090                             4518+DIMLMS   DS     CL8 .                   LAST MESSAGE SENDER
 000098                             4519+DIMTEMP  DS     CL8 .                   TEMPORARY
 0000A0                             4520+         DS     0D
 0000A0                             4521+DIMLEN   EQU    *-DIMAS .               (LENGTH)
                                    4522          END
```

```
                                        CROSS-REFERENCE
SYMBOL     LEN  VALUE  DEFN     REFERENCES

ACARD        4 000174  4490     4006
ANYBACK      8 0001D0  4507     4015  4179  4183
ASEXCP       6 001342  4093     4080
ASGNUNIT     4 0012CA  4064     4090  4096
ASIN         4 00132A  4087     4076
ASOUT        4 00133A  4091     4078
CARD        80 000084  4481     4005  4035  4044  4051  4116  4118  4120  4124  4126  4130  4132  4133
CAW          4 000048  2359     3579  3681  3761
CAWSEM       4 000194  2372     3577  3679  3759
CCBCON1      4 000C10  3617     3571  3671
CMPEXCP      6 001324  4085     4079
CMPIN        6 001318  4083     4075
CMPOUT       6 00131E  4084     4077
CORE         4 0001A8  4500     4012  4063  4098  4101  4208
CORECOMP     6 0012C0  4062     4056
CORELOOP     4 0012A8  4056     4060
COREOK       4 0012C6  4063     4057
CORESIZ      4 00115C  3972     3910
CORESIZE     1 008000  2346     3908  3972  4300
CORETAB      1 001530  4235     4055  4247
CORETABS     4 00155C  4247     4054
CSW          8 000040  2358     3868  3869  3872
DIM          1 001574  4270     4088  4316
DIMAAS       4 0015C0  4294     4275
DIMAS        1 000000  4514     4282  4521
DIMLEN       1 0000A0  4521     4294
DIMLMS       8 000090  4518     4283  4289  4291
DIMLOOP      4 001596  4285     4292
DIMMSG       8 000000  4515     4285  4286  4288  4289
DIMSEM       4 0015B8  4293     4273  4280
DIMTEMP      8 000098  4519     4288  4291
ENDADATA     6 000BDE  3604     3595
ENDCARD      4 001428  4158     4121
EXCPCOMP     4 000D84  3747     3751
EXCPDONE     4 000E00  3781     3776
EXCPFIND     2 000D9A  3753     3748
EXCPHAAS     4 000E14  3786     3727
EXCPHAS      1 000000  4459     3736  4467
EXCPHM       8 000018  4463     3766  3767  3768  3769
EXCPHMSG     8 000000  4460     3737  3768  3771
EXCPHNDL     1 000D44  3721     4093  4317
EXCPHSEM     4 000E0C  3785     3725  3732
EXCPLOOP     4 000D62  3737     3742  3783
EXCPWAIT     4 000DC0  3764     3778
EXPLOOP      6 001488  4189     4197
EXPNXT       2 00149E  4195     4192
EXPUNGE      4 001480  4186     4061  4068  4081
EXTHANDL     1 00027A  2389     2363
EXTHRET      4 0002A8  2405     2394  2398
EXTNEW       1 000058  2363
EXTOLD       8 000018  2353     2393  2401  2406
FSB          1 000000  4380     2790  2876  2945  2954
FSBNEXT      4 000000  4381     2803  2804  2811  2879  2893  2894  2948  2949  2955
FSBPTR       4 000180  2369     2786  2787  2874  2875  2941  2942
FSBSEM       4 000184  2370     2784  2830  2872  2907
```

```
                                                    CROSS-REFERENCE
SYMBOL      LEN  VALUE  DEFN     REFERENCES

FSBSIZE       4 000004  4382     2799  2812  2883  2886  2891  2946  2956
GWINC         4 000586  2672     2669
GWLOOP        4 000576  2668     2674
GWRUN         6 0005A0  2679     2671
IDLE          1 000598  2677     2675
INSEQ         8 001518  4231     4020
IOBACK        4 000F66  3889     3866
IOCOMP        6 000F02  3861     3865
IODEVFND      6 000F1C  3868     3862
IOHANDL       1 000EF4  3855     2367
IOHSAVE       4 0001DC  2375     3856  3883  3889
IONEW         1 000078  2367     2662
IONOPPR       2 000F64  3888     3875
IOOLD         8 000038  2357     3859  3861  3882  3890
IOWAIT        4 000F5C  3885     3879
IPLAPCBS      4 001150  3969     3917
IPLCCW1       8 000008  2351
IPLCCW2       8 000010  2352
IPLCL         4 000F8A  3911     3914
IPLLOOP       4 000F9E  3917     3942
IPLPCB        8 001008  3959     3905  3960
IPLPSW        1 000000  2350
IPLRTN        1 000F6E  3902     2350
IPLTH         2 000F98  3915     3912
JOB           4 00126C  4041     4036
JOBBIT        1 00007A  4444     3537  3551  3556  3608
JSP           1 0011B4  3989     3980
JSPAAS        4 001568  4249     3994
JSPAS         1 000000  4478     3999  4512
JSPNEVER      4 001500  4227     4038
JSPSUSEM      4 001560  4248     3992  4000
KEY           4 00018C  4495     4017  4094  4102  4165
KEYTAB        1 000FF0  3951     3939
LENEXCPA      1 000030  4467     3786
LENJSPAS      1 0001E0  4512     4249
LENMSG        1 00000C  4393     3287  3341
LENPCB        1 000148  4355     2991  3070  3795  3969
LENPRTHA      1 000030  4457     3711
LENRDRHA      1 000080  4446     3619
LINE        132 000000  4479     4010  4042  4043  4043  4044  4174  4175  4175  4176
LOAD          4 001352  4098     4066
LOADCL        4 0014B6  4203     4207
LOADD         4 0014C6  4208     4205
LOADED        1 0001DD  4510     4041  4100  4198
LOADLOOP      4 00137C  4111     4108  4122  4129  4149
LOADSK        4 00136C  4106     4110
LOOP          4 00123C  4028     4037  4199  4210
MCHKNEW       1 000070  2366
MCHKOLD       8 000030  2356
MEMORY        4 00018C  2371     2807  2902
MSG           1 000000  4388     3053  3264  3353  3361  4393
MSGNEXT       4 000004  4390     3054  3265  3356  3357  3362
MSGSENDR      4 000000  4389     3282  3363
MSGSIZE       4 000008  4391     3055  3273  3275  3286  3365
MSGTEXT       1 00000C  4392     3298  3376
```

```
                                        CROSS-REFERENCE
SYMBOL     LEN  VALUE  DEFN    REFERENCES

NEXTTRY      4 000274  2380    2638  2665  2679  3907
NEXTTRYM     1 000278  2381    2636  2639  2680  3885
NOTALGND     6 00140A  4150    4137
OUTSEQ       8 001524  4233    4024
PCB          1 000000  4333    2396  2436  2531  2532  2533  2587  2594  2633  2667  2687  2996  3039  3044  3048  3076
                               3131  3134  3138  3142  3145  3150  3170  3174  3177  3180  3184  3187  3191  3217  3227
                               3284  3300  3336  3378  3412  3423  3462  3474  3877  3923  3927  3931  4161  4188  4355
PCBAADDR     4 000048  4350    2835  3064
PCBASIZE     4 000044  4349    2834  3066
PCBBLOKT     1 000019  4340    2397  2597  2635  2668  2999  2999  3878  4162  4169
PCBFM        4 00002C  4345    3050  3263  3265  3351  3352
PCBFSA      84 0000A0  4352    2532
PCBINSMC     1 00001A  4341    2706  2708  2731  2733  3463
PCBISA      84 00004C  4351    2399  2531  2598  2682  2999  3413  3880  3932  4166  4167  4168
PCBLPALL     4 000014  4338    3132  3135  3171  3178
PCBLPTG      4 00000C  4336    3143  3146  3181  3188  3925  3928
PCBMSA      84 0000F4  4353    2533
PCBMSC       8 00001C  4343    3261  3349
PCBMSR       8 000024  4344    3258  3371
PCBNAME      8 000000  4334    2997  3219  3285  3449  4189
PCBNPALL     4 000010  4337    2672  2679  3128  3129  3136  3172  3175
PCBNPTG      4 000008  4335    3139  3140  3147  3182  3185  3218  3924  3929  4190
PCBNSW       4 000030  4346    2590  2591  2596  2634
PCBSES       8 00003C  4348    2741
PCBSRS       8 000034  4347    2739  3469
PCBSTOPT     1 000018  4339    2670  2998  3040  3418  3465
PCBSW        1 00001B  4342    2736  2738  3468
PGMHANDL     1 0002B0  2408    2365
PGMNEW       1 000068  2365
PGMOLD       8 000028  2355
PROGRAM      1 000000  2344    3908  4300
PROTCON1     4 000C14  3618    3562  3567  3662  3667
PRTHAAS      4 000D38  3711    3635
PRTHANDL     1 000C24  3629    4234
PRTHAS       1 000000  4448    3644  4457
PRTHCCB      4 000000  4449    3645  3672  3673  3674  3676
PRTHCOMM     4 000CBC  3679    3675  3707
PRTHLOOP     4 000C46  3646    3655  3703
PRTHM        8 00001C  4453    3694  3696  3697  3698  3701
PRTHMSG      8 000008  4450    3646  3659  3698
PRTHNO       6 000CF6  3694    3665  3670
PRTHOK       6 000D00  3696    3693
PRTHPOK      4 000C9A  3671    3660
PRTHPRIN     4 000C6E  3657    3652
PRTHSEM      4 000D30  3710    3633  3640
PRTHSEND     6 000D06  3697    3695
PRTHSTC1     6 000CB0  3676    3654
PRTHWAIT     4 000CE0  3688    3691
PTSTATUS     2 000D22  3704    3686
QUANTUM      4 0005BC  2685    2683
RDRHAAS      4 000C18  3619    3527
RDRHANDL     1 000AC6  3521    4232
RDRHAS       1 000C00  4435    3536  4446
RDRHCCB      4 000000  4436    3538  3572  3573  3574
RDRHLOOP     4 000AEC  3539    3544  3603
```

```
                                                    CROSS-REFERENCE
SYMBOL       LEN  VALUE   DEFN     REFERENCES

RDRHM          8 00006C   4441     3590  3596  3597  3598  3601  3604
RDRHMORE       4 000B2E   3559     3552
RDRHMSG        8 000008   4437     3539  3549  3559  3592  3598
RDRHNO         6 000BA0   3590     3554  3565  3570
RDRHOK         4 000BAA   3592     3589
RDRHPOK        4 000B54   3571     3560  3613
RDRHSEM        4 000C08   3616     3525  3532
RDRHSEND       6 000BC2   3597     3591  3609
RDRHSOK        6 000BBC   3596     3557  3593
RDRHTEMP      80 00001C   4440     3555  3605
RDRHWAIT       4 000B8A   3584     3587
RDSTATUS       2 000BFC   3610     3582
REGS           1 000000   4362     3935
REG0           4 000000   4363
REG1           4 000004   4364
REG10          4 000028   4373
REG11          4 00002C   4374
REG12          4 000030   4375
REG13          4 000034   4376
REG14          4 000038   4377
REG15          4 00003C   4378
REG2           4 000008   4365
REG3           4 00000C   4366     3938
REG4           4 000010   4367     3939
REG5           4 000014   4368
REG6           4 000018   4369
REG7           4 00001C   4370
REG8           4 000020   4371
REG9           4 000024   4372
REPLY        132 0000E0   4485     4033
RETURN         1 0004D8   2542     2583  2630  2640  2684  2709  2743  2837  2910  2957  3002  3073  3149  3190  3225  3296
                                   3374  3419  3467
RETURNR        1 0004E0   2544     2542
RLDCARD        4 0013CA   4132     4119
RLDCONT        4 0013EA   4141     4156
RLDFINI        2 0013FE   4146     4144
RLDLOOP        4 0013D2   4134     4148
RLDTEMP        4 0001B4   4503     4016  4150  4151  4153  4154  4155
RREPLY         8 0000D4   4483     4031  4047  4114
RREPLY1        4 0000DC   4484     4030  4113
RUNNING        4 000270   2379     2395  2435  2595  2681  3876  3906  4186  4196
SA             1 000000   4357     2400  2444  3414  3415  3417  3422  3891  3933
SAPSW          8 000000   4358     2401  2445  2545  3415  3415  3416  3882
SAREGS        64 000008   4359     2402  2446  2546  3417  3417  3883  3934
SATEMP        12 000048   4360     2813  2896  2982  3031  3047  3068  3329  3404  3453
SCAN           2 0014D0   4212     4052  4064  4074
SCANLOOP       4 0014D2   4213     4221
SEQ            8 00019C   4498     4069  4070  4073  4082  4187  4189  4191
SETDIM         6 00132E   4088     4092
SHORT          4 0013FA   4145     4142
SKIP           8 001508   4228     4008  4181
SM             1 000000   4384     2577  2625  2901
SMPTR          4 000004   4386     2584  2585  2631  2634
SMVAL          4 000000   4385     2578  2580  2626  2628  2904
STOP           4 001266   4038     4034
```

```
                                   CROSS-REFERENCE
SYMBOL     LEN  VALUE  DEFN    REFERENCES

STREAMS      4 000FE4  3945    3916
SVCCONST     4 000314  2455    2429
SVCHANDL     1 0002B2  2425    2364
SVCHPROT     4 000302  2449    2434
SVCHTABL     1 000328  2457    2431  2458  2460  2462  2464  2466  2468  2470  2472  2474  2476  2478  2480  2482  2484
                               2486  2488  2490  2492  2494  2496  2498
SVCNEW       1 000060  2364
SVCOK        4 0002D0  2435    2451  2453
SVCOLD       8 000020  2354    2430  2442  2445  2449  2545  2547
SVCRTN       8 000428  2500    2432  2452
SVCSAVE      4 0004C8  2529    2441
SVCXPER      4 0002FA  2447    2443
SYSSEM       4 0002DE  2440    2438
SYSSEMSA    84 00021C  2377    2530  2598
TALK         8 0001B8  4504    4013  4014  4171  4176
TEMPLATE     4 001178  3977    2999
TIMER        4 000050  2361    2683
TOKSTART     2 0014F6  4222    4215  4217  4219
TRAPSAVE     4 00019C  2374    2390  2402  2405  2426  2446
TREAD        4 000164  4486    4002  4003  4004  4028  4111
TXTCARD      4 0013B0  4124    4117
TXTMOV       6 0013C4  4130    4128
TYPLEN       1 000054  3981    3921
TYPPCB       8 001160  3975    3921  3981
UCB          1 000000  4469    3522  3630  3722  3867  4476
UCBADDR      4 000000  4470    3580  3684
UCBCSW       8 000014  4473    3575  3576  3586  3588  3682  3683  3690  3692  3757  3758  3766  3868  3870  3871  3872
UCBFPR       1 00001C  4474    3874
UCBLENG      1 000020  4476    3749  3863
UCBLP1       4 000FF8  3955    3948
UCBLP2       4 001000  3956    3949
UCBPRT1      4 000E94  3827    3955
UCBPRT2      4 000ED4  3841    3956
UCBRDR1      4 000E74  3820    3955
UCBRDR2      4 000EB4  3834    3956
UCBTAB       1 000FE8  3947    3936  3939
UCBTABLE     4 000E74  3818    3746  3860  4305
UCBTBEND     1 000EF4  3847    3750  3806  3864  4306
UCBUS        8 000004  4471    3547  3599  3657  3677  3699  3754  3781
UCBWS        8 00000C  4472    3584  3611  3688  3705  3764  3873
UNAMMOV      6 001312  4082    4071
UNITRTN      4 0001A4  4499    4088  4093
UNUSED0      4 00004C  2360
UNUSED1      4 000054  2362
USERL        8 000190  4496    4007  4049  4158  4160  4163
VERYEND      8 001648  4325    2369  3908  3908  4300
WRITE        8 000178  4491    4008  4009  4011  4045  4177
XA           1 000600  2771    2513  2778  2779  3793
XABACK       2 0006B0  2836    2833
XACOM        2 00060E  2780    2774
XAFOUND      4 000662  2810    2802
XALOOP       2 00062A  2791    2805
XANF         2 000686  2821    2817
XARETURN     4 00069C  2830    2825
XATOP        4 000616  2784    2809
```

```
                                            CROSS-REFERENCE
SYMBOL     LEN  VALUE  DEFN    REFERENCES

XAUTO        1 000608  2775    2527
XAWAIT       2 000656  2806    2792
XAX          1 000000  4395    2782  2990  3338  3528  3636  3728  3918  3995  4276
XAXADDR      4 000004  4397    2810  2994  3347  3530  3638  3730  3920  3997  4278
XAXALGN      4 000008  4398    2788  2992  3345
XAXSIZE      4 000000  4396    2783  2991  3344
XB           1 000744  2936    2512
XBINSERT     4 000770  2952    2944  2947
XBLOOP       4 00075A  2946    2951
XBX          1 000000  4404    2814  2897  2938
XBXADDR      4 000004  4406    2818  2826  2899  2940
XBXSIZE      4 000000  4405    2819  2827  2898  2939
XC           1 000780  2978    2520
XCERR        2 0007C4  3003    2987
XCOM         1 0005D2  2728    2511
XCOMRET      4 0005FC  2743    2735  2737
XCX          1 000000  4408    2981
XCXNAME      8 000000  4409    2984  2997
XD           1 0007C6  3027    2524
XDCHECK      6 00081A  3064    3052
XDERR        2 000840  3074    3038  3041
XDLOOP       2 0007F4  3051    3063
XDTHEN       4 00082E  3069    3065
XDX          1 000000  4411    3030
XDXNAME      8 000000  4412    3033
XEXC         1 0005C0  2703    2510
XF           1 0006B6  2863    2514
XFBACKUP     2 000706  2892    2884
XFINC        4 000708  2893    2888
XFLINK       4 000714  2896    2878
XFLOOP       2 0006D4  2877    2895
XFTHEN       2 0006F0  2885    2881
XFVDO        2 00073E  2911    2906
XFVLOOP      4 00072E  2906    2912
XFX          1 000000  4400    2867  3058  3291
XFXADDR      4 000004  4402    2869  3059  3069  3292
XFXSIZE      4 000000  4401    2868  3060  3070  3293
XH           1 000842  3096    2526
XHLOOP       4 000848  3100    3102
XHMSG1       8 000854  3104    3098
XHMSG2       8 00086C  3107    3100
XI           1 00087A  3126    2515
XJ           1 0008A6  3167    2516
XN           1 0008CA  3212    2521
XNFOUND      4 0008E4  3224    3220
XNLOOP       4 0008CC  3218    3222
XNX          1 000000  4414    2983  3032  3214  3330  3405  3454
XNXADDR      4 000008  4416    2986  3035  3224  3333  3408  3457
XNXNAME      8 000000  4415    2984  3033  3219  3331  3406  3455
XP           1 0004EE  2575    2508
XPER         1 00056A  2661    2403  2517  2599  2677
XPLOOP       2 00050A  2588    2592
XPTHEN       6 00051C  2595    2589
XPWAIT       4 000502  2584    2582
XQUE         1 000A8E  3493    2525
```

```
                                        CROSS-REFERENCE

SYMBOL     LEN  VALUE  DEFN     REFERENCES

XQUELOOP     4 000A94  3497     3499
XQUEM1       8 000AA0  3501     3495
XQUEM2       8 000AB8  3504     3497
XR           1 0008EC  3254     2518
XRAFT        4 000938  3280     3278
XRFILL       6 00096C  3297     3271
XRMOVE       6 000972  3298     3279
XRNOB        4 00091C  3272     3270
XRTHEN       2 00092E  3277     3274
XRX          1 000000  4418     3257  3540  3550  3647  3738  3772
XRXNAME      8 000000  4419     3285  3553
XRXSIZE      4 000008  4420     3267  3281  3541  3648  3739  3773
XRXTEXT      1 00000C  4421     3269  3297  3297  3298  3543  3545  3650  3651  3653  3741  3743  3744  3775  3777
XS           1 000978  3325     2519
XSADD        4 0009D2  3359     3355
XSAFT        2 0009F4  3370     3368
XSERR        2 000A02  3375     3335
XSLOOP       2 0009C0  3354     3358
XSMOVE       6 000A04  3376     3369
XSX          1 000000  4423     3328
XSXNAME      8 000000  4424     3331
XSXSIZE      4 000008  4425     3340  3364
XSXTEXT      1 00000C  4426     3376
XV           1 000534  2623     2509
XVRET        4 000566  2640     2637
XVWAKEUP     4 000548  2631     2629
XY           1 000A0A  3400     2522
XYERR        2 000A40  3420     3410
XYX          1 000000  4428     3403
XYXADDR      4 000008  4430     3416
XYXNAME      8 000000  4429     3406
XZ           1 000A42  3445     2523
XZERR        2 000A8C  3472     3452  3459
XZFINE       4 000A54  3453     3450
XZINSMC      4 000A7E  3468     3464
XZSTOP       4 000A6C  3463     3471
XZX          1 000000  4432     3448
XZXNAME      8 000000  4433     3451  3455


NO STATEMENTS FLAGGED IN THIS ASSEMBLY


 2332 SYSIN SOURCE RECORDS
 4984 SYSPRINT OUTPUT RECORDS
```

演 習 問 題

演習問題は次の4つに区分されている：

1. 一般的な問題
2. 小さな欠点を訂正する問題
3. 現在のモジュールに大きな変更を施す問題
4. 現在のオペレーティング・システムに大きな追加を行なう問題

一般的な問題

7.1 利用者ジョブが2つ実行していて，それぞれ1台のカード読取り機と1台の印刷機を使用しているとすると，例題オペレーティング・システムには幾つのプロセスが存在しているか．

7.2 割込みを不能にすること（マスクすること）と“システムは完了しなければならない（SMC)”という基本機能を利用することの相違点は何か．

7.3 プロセスがすべてCPU限界のものであるとき，プロセス・スケジューラのアルゴリズムはすべての利用者ジョブに対して公平か．（利用者プロセスの1つが計算を実行するのに10個の副プロセスを創成したような場合を考えてみよ．）どのようにしたらプロセス・スケジューリングを“もっと公平”とすることができるか．

7.4 作業域割付け機能（SVC C'E'）は現在のところオペレーティング・システムのプロセスに限定されている．利用者プロセスがこの機能を使用することを認めてよいか．その場合どのような影響があるか．

7.5 利用者プロセスが，PおよびVオペレータと同じ機能を果たすために，メッセージ送信およびメッセージ受信機能をいかに使用すればよいかを説明せよ．

7.6 例題オペレーティング・システムがプロセス管理で2つのレベルを有している理由は何か．

7.7

1. 例題オペレーティング・システムでは割付けおよび解放のSVC（記憶管理）は利用者から呼出すことができない（これらはオペレーティン

グ・システムのルーチンに限定されている)．この制限が設けられた理由は何か．

(ヒント：すくみの発生する可能性を考えてみよ．)

2. 上の問題を解決することができるか．すなわち，問(1)での問題を引き起こさないようにして，記憶管理を利用者が呼出せるようにすることができるか．どのような制限，条件を課す必要があるか．

3. 記憶管理は，PCB やメッセージ制御ブロックなどを作るために，オペレーティング・システムの内部で使用される．このような内部的な使用ですくみが発生するか．簡単な例を示せ．

4. 問 3. で示した例に対する解決法を提案せよ．

部分的な動作上の欠陥を直したり部分的な改善を行なう問題

7.8 停止状態のプロセス：　現在のところ，“プロセス停止”基本機能(ルーチン XZ)を用いた後に，停止したこのプロセスを“プロセス起動”基本機能(ルーチン XY)を用いて再起動させることができない．この理由は，停止状態のプロセスは“SMC 部分から出る”という基本機能(ルーチン XCOM)の中に入っており，他のプロセスがアクセスすることのないセマフォアを待合せているからである．“プロセス起動”ルーチンの中でこのような条件に対する判定を行なってみよ．

7.9 利用者セマフォア：　現在のところ，セマフォアに関するPおよびV動作は利用者プロセスから呼出すことができない．この理由は，ある種のシステム情報(たとえば，このセマフォアを待合せている最初のプロセスの PCB へのポインタ)はセマフォア自体に格納されるからである．したがって，このセマフォアが利用者の区画の中に格納されると，ポインタの正当性が保証できなくなる．このようなわけで，現在のところ，システム・ルーチンに限ってPおよびVルーチンを呼出すことが許されている．そして，これらのルーチンを呼出したりルーチンは，システム・セマフォア(利用者の区割の外に格納されており，システム・ルーチンだけがアクセスできる)を引数として引渡すことだけが可能である．プログラムで，セマファア型の同期を行なおうとするときに利用できる機能を提案せよ．基本的には，この機能は，利用者の要求によってシステム領域内にセマフォアを作るための入口と，それ以後の擬似PおよびV要求で使用するために作ったセマフォアの名前を返却する入口とを持つ必要がある．この問題を解

く主な利益は，PおよびV機能および記憶域割付けの問題点を一層よく理解できることにある．利用者プロセスによって割付けられたセマフォアとセマフォアの名前の記録を完全に残しておき，これらをジョブの終了時に解放できるようにしなければならない．この記録は，読者が詳細を迅速に理解するのに役立つであろう．

7.10 自動記憶域： 現在のところ，作業域割付け機能（SVC C'E'）は，プロセスごとに1つの作業域だけを割付ける．1つのプロセスで何回もSVC C'E' を用いることができるように，この機能の拡張を計れ．

7.11 メッセージ： 送信者の PCB を指すポインタが，受信者の PCB に対するメッセージの鎖に格納されている．受信者プロセスがメッセージを読取るより前に送信者プロセスが破壊されたならばどのような誤りが発生するか．この問題を避けるにはどのようにすればよいか．

7.12 会計情報： 現在組込まれているシステムでは課金が行なわれていない．$JOB カードに "USER=利用者名" の形式の欄が置かれているようにシステムを変更せよ．ここで，利用者名はこの計算の料金を請求すべき利用者の名前である．ジョブの終了時に，利用者名と使用した資源，および合計の経費を含んだ料金票を印刷する．この問題では，読者は料金体系をも決定せよ．ジョブが利用した資源に関する情報（たとえば，処理装置時間）を収集する必要があることに留意せよ．

既存のモジュールを大幅に変更する問題

本節の問題は，現在のシステムの構造をほとんど変更しないが，システムの機能には大きな影響を与え，教育的にも重要な内容を有している．

7.13 すくみの回復： 現在のところ，実行中のプロセスが，記憶管理が用意できる以上の記憶域を要求し，記憶域を解放するプロセスが存在しない場合にはすくみが起こる．これが起こるとシステムが完全に停止しない場合でも，システム能性が極端に低下することがある．オペレーティング・システムは，すべてのプロセスが閉塞され実行可能となり得るプロセスが存在しないことを判定できなければならない（入出力動作は行なわれていない）．トラフィック制御によって行なわれるこの判定によって，システムから追放すべきジョブが選択される．すくみ状態の検出は容易であるが，システムを変更すると問題が発生する．現在のところ，システムからジョ

ブを追放するルーチンが記憶割付けを要求するメッセージを送信する．要求する記憶域はただの数語であるが，このような状況が起こると，この要求を満たすことすらも不能となる．教育上で最も重要な問題点は，システムの最も基本的な部分であるトラフィック制御がこの問題からずっと離れたところにあるということである．これは機能単位という点からはきわめて重要な問題であり，これをいかに解決したらよいかはあまり明白ではない．

7.14　すくみの回避：記憶管理のすくみに関する別の問題は，部分的なすくみの検出である．5つのジョブ流がシステムにあり，その内4つはすくみの状態となっており，残りの1つはさしあたり記憶割付けを行なわないプログラムを用いてCPU限界で長時間実行しているような場合が考えられる．CPU限界の部分を完了した後に，このジョブは他のジョブを実行するのに十分な記憶域を解放することもあるが，他のジョブと同じすくみの状態となってしまうこともあり得る．重要ではあるが難しい割当て法は，トラフィック制御でこの種のすくみを検出できるように機能の拡充を行ない，この問題を前の問題と同様に扱うことである．

7.15　装置割付け：　現在のところ，各ジョブは1台の読取り機と1台の印刷機とを要求することができ，いつこの装置が割当てられるかは，この要求が出される前に決定される．これは確かに最も効果的な方法ではない．特に読取り機より印刷機の台数が多いときにはより効果的な方法が考えられるが，単純で，装置要求のすくみの問題を解消することができる．ジョブが使用する読取り機にジョブのカードがなくなれば，これを解放して，次のジョブの処理を始めるようにスーパバイザ・プロセスを作ることができると，システム性能の改善を行なうことができる．

既存のシステムに大幅な追加を行なう問題

本節の問題は，システムに新しく主要な機能を追加しようとするものである．これらの機能は種々あり，それらの代表的なものの幾つかが下にて論じられる．

7.16　2次記憶装置の利用：　ディスクなどの2次記憶装置を処理するルーチンをシステムに追加せよ．この変更は，ジョブ流処理と装置管理ルーチンに必要であろう．

7.17 ファイル・システムの設計： ディスク管理が使用できるようになると，ファイル・システムを追加することが可能となる．これを組込む方法は種々存在するが，利用者からは，$JOB カードの欄を用いることが容易であることを留意せよ．

7.18 スプーリングとジョブ・スケジューリング： ディスク管理と簡単なファイル・システムがあると，入力および出力スプーリングを行なうことが可能となる．そして，スプーリングを行なうと，各種のジョブ・スケジューリング・アルゴリズムを追加することが可能となる．

第 8 章

相互の関係：性能評価

本章の目的は次の3つである：

1. 資源の間の相互関係を読者に“気づいて”もらうこと．
2. 相互の関係のモデル化と解析法を紹介し，文献検索の糸口を与えること．
3. システムが構築されたり構成されたりする前に，相互の関係を解析しておくことが有意義であることを示すこと．この目的のために，2つの場合について解析してみる．

相互関係の範囲と組合せは無限であるから，この問題をすべて論じることはできない．しかし読者が直面するかもしれない基本的な質問の幾つかを答えるのに役立つような手法を示すことはできる．たとえば，読者が1台の計算機システムを持っていて，読者の上司がシステム性能改善のために一定額の金を支出するとしたら，記憶装置を増設するか．もっと速い CPU を買うか．ディスクを買うか．チャネルを増設するか．本章を読んだ後に，この問題に対する答えを出すための一般的な考え方を修得したり，文献の利用法を修得したり，数学的解析を行ない得るようになったりすることを期待したい．

1つの資源管理が他に及ぼす影響を説明してみよう．記憶装置の高い利用率を達成させる記憶管理方式，たとえばページングと，処理装置の利用率を高め待合せ時間を短かくする最良の多重プログラミングの処理装置管理方式とを使用すると，最もよいシステムを得ることができると考えるであろう．これは必ずしも真ではない．これら2つの方法は，処理装置が利用者プロセスの実行を行なわずに大部分の時間をオーバヘッドの機能を実行するスラッシングという現象を起こすような関係になっている場合もある．

本章では各資源管理にどのような相互関係が含まれるか順に論じ，次に利用

者が制御し得るものの幾つかについて述べる．次いで，処理装置管理と記憶管理との相互関係の2つの面——スワッピングとページング・システム——の解析を行なおう．スラッシングについても解析を行なう．

相互関係の詳細については，註釈をつけた文献を参照してほしい（相互関係，モデリングの項参照）．スラッシングの形式的な解析は，**マルコフ・モデリング**と呼ばれるモデル化技法を利用する（Feller, 1969；Drake, 1967）．この解析法は有用であり，その応用例は文献にも示されている．文献表には，システム動作のモデル化を論じている20編の文献と，性能評価と測定とを扱った6編の文献とを掲げている．ページング性能の種々の側面を解析した文献は19編以上存在する．これらの参考文献は，本章の議論を拡張するのに用いることができる．

8.1 記 憶 管 理

第3章では一連の記憶管理技法について論じた．これらによって記憶装置の利用率が順番に向上した（単一連続割付け，分割，再配置型分割など）．しかし，記憶装置の利用率の改善には CPU 時間が余分に必要であった．

記憶管理技法を選択する前に，記憶装置と CPU 時間との関係を解析しておくことが必要である．例として，図 8.1 に示した再配置型分割割付け方式の各区

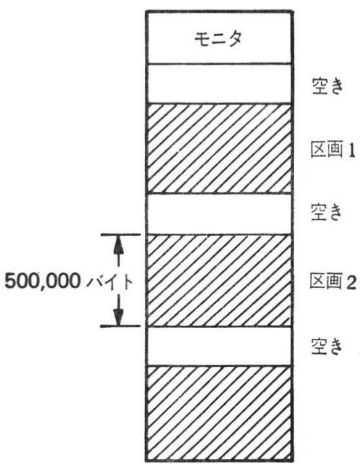


図 8.1　再配置分割方式

画の大きさが500,000バイトであると仮定しよう．“転送”速度が毎秒4,000,000バイトとすると，1つの区割を移動させるのに125ミリ秒かかる．システムの中の大多数のジョブが学生のジョブであれば，それを実行するより移動させる方が長い時間かかる．学生のジョブの実行時間はただの100ミリ秒程度である(100,000命令の実行)．

8.2 処理装置管理

第4章では処理装置管理を論じ，多重プログラミングは処理装置の利用率を向上させるが，記憶装置と入出力装置の負荷が増すことを示した．

第3.2.2節では，CPUの入出力待合せ率を多重プログラミングするジョブ数と各ジョブを単独で実行したときの入出力待合せ率の関数として計算した．

図8.2は，入出力を待合せているCPU時間（$T_{I/O}$）をシステム中で多重プログラミングされているジョブ数の関数として示した曲線である．ここではすべてのジョブは等しく，単独に実行すると同一量の入出力待合せ時間を生じるものと仮定している．特定のジョブ群の入出力待合せ率に関するデータは表3.1に示されている．

図8.2は，$T_{I/O}$が0に近ずくと（多重プログラミングの多重度が高くなること），CPUが完全に利用されることを示している．すなわち，CPUはたえず利用者のために有用な仕事を実行している．しかし，多重プログラミングのジョブ数を増すとオーバヘッドが増して，利用者プロセスの実行のためのCPU時間が減少し始める点が存在する．

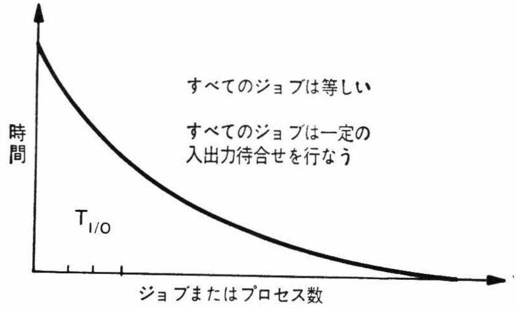


図 8.2　ジョブ数と入出力待合せ時間の関係

このオーバヘッドは次の2つの要素から生じる：

1. 入出力の行列を管理するためにCPU時間が必要である．この時間は通常一定の時間であるが（たとえば，1ミリ秒），ジョブ数が増すとオーバヘッド時間が増す．
2. チャネル，制御装置，および入出力装置の台数は限定されている．この結果，競合が発生する（たとえば，同一のディスクに8つの要求が出され，第1の要求は100ミリ秒で満たされるが，最後のものは800ミリ秒経っても満たされない）．したがって，プロセス数が増加するとプロセスが動作中の装置を待合せてCPUがアイドルとなる時間が長くなる．

図8.3は，使用できないCPU時間の合計が示されている（入出力待合せ時間と行列を管理するオーバヘッド）．多重プログラミングの多重度が一定値を越えるとシステムの性能は低下することに留意してほしい．（この例はページングを行なっていないシステムのものである．ページングを行なうシステムで多重プログラミングの多重度を上げすぎると性能の低下はもっと著しい．）置与えられたシステムの多重プログラミングの多重度には，これを越すと記憶装と入出力装置の双方に不利になる点が存在する．

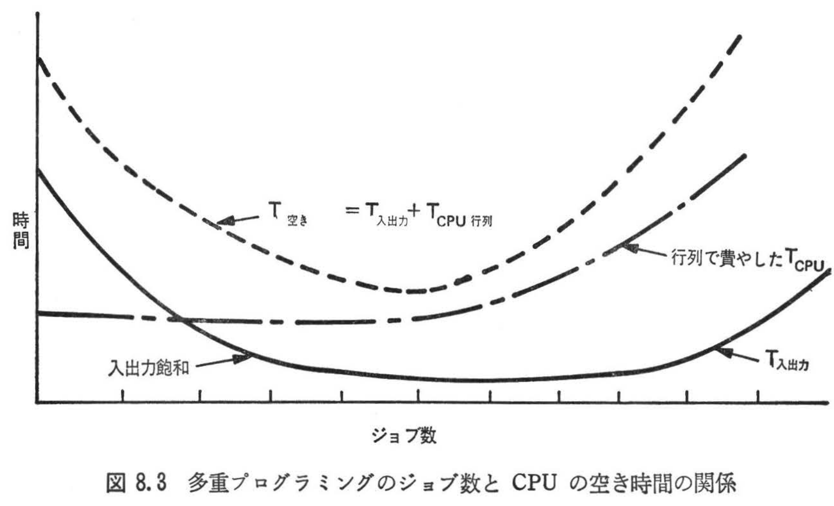


図 8.3 多重プログラミングのジョブ数と CPU の空き時間の関係

8.3 装置管理

第5章では装置管理を改善するための幾つかの技法を論じた（たとえば，行列の順番変更，バッファリング，ブロッキング，スプーリング）．これらの技法はシステム性能を改善するのにどのように役立つたであろうか．多くのシステムでは，これらの複雑な技法によって効率が向上するよりも多くのオーバヘッドの増加を紹くことがある（Teorey, 1972）．それぞれ25％の入出力待合せを行なう2つのジョブが多重プログラミングされているシステムを例として考えてみよう．CPU がアイドルとなるのはどれだけの時間であろうか．表 3.1 からは，CPU 時間の4％がアイドルになることがわかる．このようなシステムで，最上の装置管理を行なうことはどれだけの節約となるか．多重プログラミングによって入出力待合せ時間が4％に減少しているので，この値を0にすることだけしか期待できない．したがって，オーバヘッドが CPU 時間の4％以上であるような装置管理は，効率を上げる以上のオーバヘッドを生じてしまう．

しかし，多数のジョブは25％以上の入出力待合せ時間を持つのであるからすべてのシステムについて，上述の議論が成立すると結論すべきではない．

本節では，次の技法の使用に関する相互の関係を調べてみよう：

行列の並べ換え
ブロッキング／バッファリング
データ詰め合せ技法

8.3.1 行列の並べ換え

行列が適当でない例を幾つか示そう．CPU と命令実行の速度の範囲を考えてみる（図 8.4 参照）．次に幾つかのディスクとそれらのアクセス速度とを見てみよう（図 8.5 参照）．アクセス時間は入出力起動命令実行時からその装置の入出力完了時までの時間であり，この間 CPU は待合せなければならないことがある．たとえば，IBM 2314 のアクセス時間は75ミリ秒であり，この値は CPU がアイドルとなる最大の時間である．実際，多重プログラミングを行なっている場合には，他のプロセスを実行するので，このような長い時間待合せることはない．

行列管理ルーチンには 1,000 命令もの実行が行なわれるものがある．IBM

計算機	命令の実行時間 (マイクロ秒)	1000 命令の実行時間 (ミリ秒)
360/30	30	30
360/40	10	10
360/50	3	3
360/65	1	1
370/158	0.4	0.4
370/168	0.2	0.2

図 8.4 計算機の命令実行速度

	ディスク	アクセス速度 (ミリ秒)
可動腕型	IBM 2311	125
	IBM 2314	75
	IBM 3330	35
ドラム型	IBM 2301	10
	IBM 2305	5

図 8.5 ディスクの速度

360/30 では，この時間は IBM 2301 や 2305 を乱アクセスするより長く，IBM 2314 をアクセスする時間に近い．IBM 360/65 で入出力負荷がかなりある場合には，IBM 2305 をアクセスするときでも行列に入れることに意味がある．**註**：入出力の負荷が適度であって多重プログラミングが行なわれている場合には，入出力は**すべて**同時実行が可能となることがあり，行列管理時間だけが CPU のオーバヘッドとなる．

8.3.2 ブロッキング/バッファリング

ブロッキングとバッファリングは，入出力回数を減少させて入出力装置の容量を効果的に利用するための装置管理技法である．

しかし，バッファのための記憶域とデブロッキングのための CPU 時間とが余分に必要となる．多くのシステムでは，多重プログラミングの多重度を上げると，バッファを余計に用いて入出力の能率を上げるよりも，記憶装置の利用効率が上がり，入出力装置の容量は大した問題ではない．たとえば，1つのバ

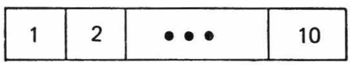
ッファが4Kとして，ダブル・バッファリングを行なうファイルを2つ（入力および出力）を用いると16Kの容量が必要である．この容量は“小”ジョブの容量以上であろう（IBM 360/30 は最大 64K までである）．

8.3.3　データ詰め合せの技法

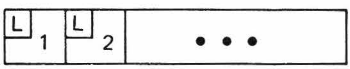
次に示す例で3つのレコード詰め合せ技法は，CPU 時間，入出力のアクセス，記憶域の相互関係を示している．例は通常のカードをスプーリングして，次に示す特性を有している：

1. 出力スプールの行は約80〜132バイトに限定される．
2. 80バイトの内約50%は空白であり，しかも連続していることが多い．

技法 1　つねに132バイトのレコードを出力する．ディスクのブロック＝1320バイトとすると，1ブロック当り10レコード入る．



技法 2　後に続く空白を切捨てて可変長のレコードを出力する（1ブロック当たり平均16レコード）．このようにして入出力時間を減少させる．



技法 3　連続する空白を符号化する（たとえば FF | n の2バイトは，n (>2) の長さの空白を示す）．このようにすると1ブロック当り20レコード入り，入出力時間が減少するが CPU 時間は増加する．

読者は次の問に答えてみてほしい：

1. 技法2は1よりも入出力時間が多いことがあるか．
2. 技法3は2に比べてどうか．
3. ブロック容量とバッファ数の関係はどうか．
   a．（実行中の利用者10人）×（入力2＋出力2）＝40バッファ
   b．（40バッファ）×（4,000バイト/バッファ）＝160,000バイト（スプーリングのためのバッファ容量が半分（80K）であればどうなるか．）

例題オペレーティング・システムでは，システムが CPU 限界であるので，

技法2が選ばれている．

8.4 情 報 管 理

第6章では，情報管理の技法を論じた．2次記憶域割付け技法は，主として，利用者のための融通性を増すことを目的として論じた（動的な割付けによって，利用者ファイルの長さが大きくなり得る）．しかし，2次記憶域自体の節約についても考慮するべきであろう．次は2次記憶域を節約する4つの技法を示している．これらの技法はCPU時間を消費する：

1. ブロッキング——論理レコードを群管理すること．
2. 多重ブロッキング——ファイルを群管理すること．このようにすると“ファイル余り”によって無駄になっている領域を解消することができる．（ファイル長が物理ブロック長の整数倍となっていることは滅多にない.）
3. 符号化——たとえば，連続した0を符号と0の個数とで置換える．
4. 意味による詰め合せ——たとえば，

   LET ALPHA＝BETA＋GAMMA

   という文は，各数字がキーワードや記号を表わすとして

   471839

   というように記憶することができる．

8.5 利用者が制御できる相互関係

次に示す3つの技法は，記憶域を増すことによって，アルゴリズムの速度を上げることができる：

1. ハッシュ表の利用（Donovan, 1972；Morris, 1968）—— 探索を要する表の管理にハッシュ符号化技法を利用して，高速のアクセスを可能とすることができる．しかし，高速のアクセスを保証するためには，ハッシュ表は疎に記入（たとえば，半分だけ記入）しておく必要がある．したがって，ハッシュ表を用いて情報を格納するには，2分探索技法の場合の2倍の領域が必要である．

2. 関数表の利用——関数を計算せずに表を用いることができる（たとえば，毎回計算する代わりに，必要となる sin の値をすべて格納しておく）.
3. 入出力の回数を減少させたりあるいは全くなくしたりする——情報を主記憶に格納することができる．これは主記憶域と入出力の直接的な関係である．

8.6 影　　響

CPU 時間や領域を節約する技法を有効に利用するには，利用者はオペレーティング・システムのアルゴリズムを理解しておく必要がある．さもなくば，"うまく整合がとれず"，実際にはシステムの性能を低下させることになるかもしれない．また相互の関係を調整しようとする資源の価格についても，もちろん，知らねばならない．

利用者は，自分自身の性質をも考慮しなければならない．たとえば，待つのがいやであれば，経費がいくらかかっても迅速な返却を要求するであろう．

8.6.1 多重プログラミング・システムの利用者に与える影響

使用するシステムが多重プログラミングを行なっているか否かを知っておくことは重要である．たとえば，システムが多重プログラミングを行なっていない場合には，主記憶が 256K で 100K のジョブを実行するということは普通である．このようなシステムでは領域の容量は，通常，問題とならない．したがって，利用者は CPU 時間を節約するために自由に領域を使用することが可能である．図 8.6 は，アルゴリズムの性能と領域の大きさ（たとえば，より多くのバッファを用いる）との関係を示している．領域がもっとたっぷりあると，利用者はハッシュ法を用いたり，関数表を用いたりして実行時間を減少させることができる．

しかし，多重プログラミングを行なっているシステムでは，1人の利用者が記憶域の使用量を増すと，システムは多くの利用者を多重プログラミング実行できず，システムの性能を低下させることになる．図 8.7 は，多重プログラミングを行なっている場合に，1ジョブ当りの主記憶容量とシステム性能（仕事量）との関係を示している．図 8.7 の点1は，システム内の各ジョブが小さな記憶域を有している場合を示している．各ジョブにもっと記憶域が与えられる

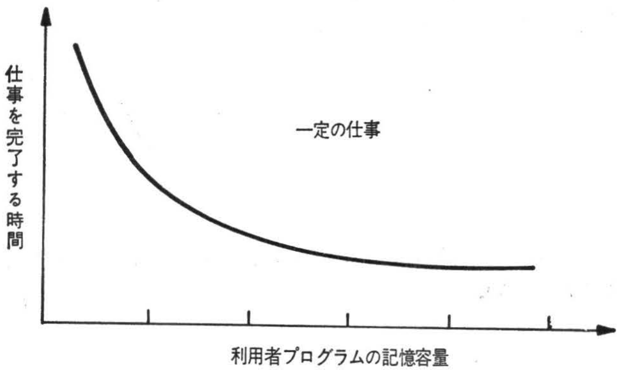


図 8.6　利用者プログラムの性質

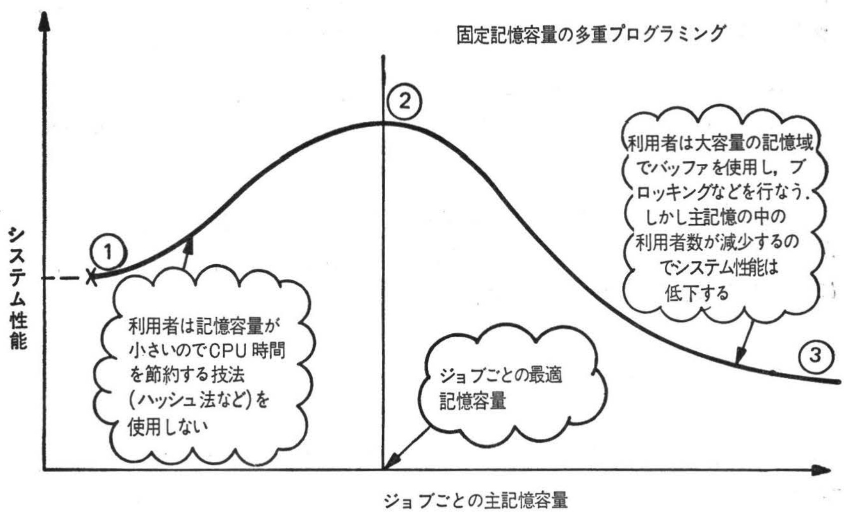


図 8.7　多重プログラミング・システムの性能と記憶容量の関係

と，たとえば入出力の回数が減少して，システムの性能が改善される（点 2）．

しかし，利用者がもっと記憶域を使用すると，他の利用者がロードできず，多重プログラミングのジョブ数が減り，システム性能が低下する（点 3）．最良の主記憶容量（1 ジョブ当り）は点 2 であるが，この点はシステム内のジョブの性質によって変化する．

利用者による入出力アクセスと CPU 時間の関係調整について，幾つかの経

費の測定が行なわれている．全負荷状態のCP-67では，CPU時間12ミリ秒が1回の入出力装置のアクセスと等価であることがわかっている．すなわち，バッファリングやブロッキングのためのCPU時間が12ミリ秒以下で，1回の入出力を節約することができれば，能率がよくなる．

利用者があまり制御できない相互関係の例であるが，CPU時間11ミリ秒と1.4回のページ入出力の経費が等しいことが示されている．この場合，CPUでの計算を増してページ要求を減少させると経費を逓減することができる．ページングは利用者にわからない（ようにすべきである）から，この比較は直ちに利用することはできない．

8.6.2　心理学的な関係

最近，著者は小さな会社の会計報告パッケージを作る仕事に関係した．この報告はアルファベット順に印刷するものであるが，データベースはアルファベット順に並んでいないので，2つのアルゴリズムが考えられる：

1. 各項目ごとにデータベースを探索して印刷する．
2. データベースを分類して，分類されたデータベースを丸ごと印刷する．

第2法はデータベースの大きさからいって効率が優れているが，著者の1人が答えが出てくるまでの間に受ける不安に耐えかねたので，能率の悪い第1法にプログラムし直したことがある．

8.7　スワッピングとページング

本節では，記憶管理と処理装置管理の相互関係を解析する．本節では次の2つの記憶管理技法を考える：

1. スワップ・システム（たとえば，TSO, CTSS, PDP-10）——スワップ・システムは，処理装置が割付けられる前に，番地空間が主記憶にロードされるような記憶管理技法を使用する．別のプロセスをアクティヴにするには，主記憶中の番地空間を2次記憶にスワップして，新しい利用者をロードする（第3.9.1参照）．
2. デマンド・ページング・システム（たとえば，TSS/360，VM 370，370 OS，CP-67，MULTICS）——第3.6節参照．

測定をしたいのは次の2つの値である：

1. システムの応答性——システムが利用者に応答する時間
2. 仕事量——各ジョブごとにかかった時間の平均値

記憶管理の2つの方式でプロセスに割当てられる処理装置時間を変化させてみよう．これはシステムの応答性に直接関係する．

8.7.1 例題のプログラムとシステム

図8.8は，例題計算機の特性を示している．例題プログラムの特性を次のように仮定する：

容量＝100K語（100ページ）
実行時間＝1秒（10^6命令）

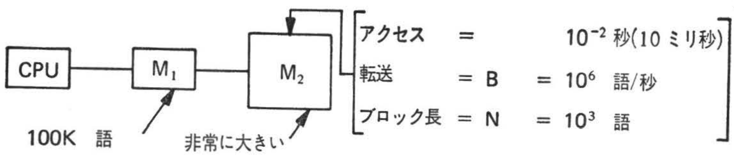


図 8.8 ページング対スワッピングの解析例で考察する計算機

目的とすることは，スワッピングとページングの記憶管理技法を用いて，図8.8のシステムで例題プログラムを実行する時間を解析することである．すなわち，図8.9の“?”の欄を求めることが目的である．

8.7.2 場合1：割当て時間＝∞

第1の場合には，割当て時間(tq)＝∞とする（すなわち，ジョブは完了するまで実行される）．このように大きな割当て時間を与えると，実行中のプロセス以外にはシステムは応答することができない．したがって，多くの利用者に応答しない時間が長くなる．

8.7.2.1 スワップ・システム

図8.10は，tq＝∞の場合に実行時間の関数として，スワッピング・システムの記憶域利用率を示したものである．例題プログラムを実行する前に，これを探索し（アクセス），ロードする必要がある．

番号	システムの応答	割当て時間	記憶管理方式	合計の実行時間
1	低速	割当て時間＝∞	スワッピング	?
			ページング	?
2	中速	割当て時間＝500ミリ秒	スワッピング	?
			ページング	?
3	高速	割当て時間＝50ミリ秒	スワッピング	?
			ページング	?

図 8.9　ページングとスワッピングの比較

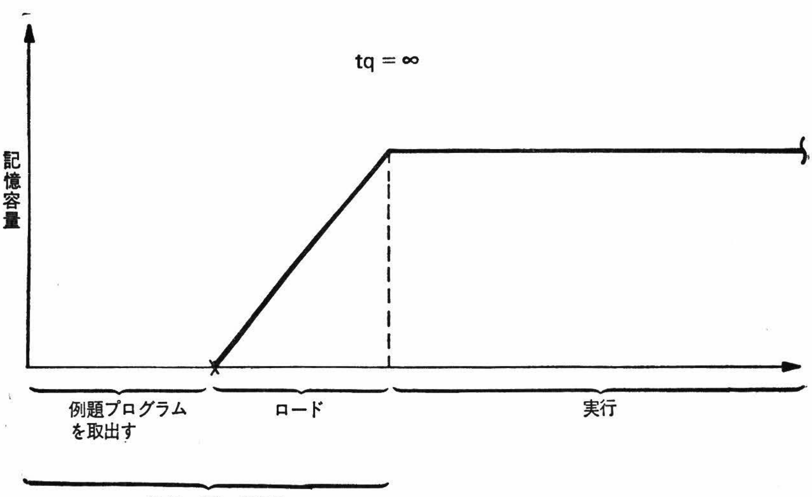


図 8.10　番号1でスワッピングする場合には例題プログラムが使用する記憶容量

図 8.10を用いて，$tq=\infty$ の場合のスワッピング・システムでの例題プログラムの全実行時間は次のように計算することができる：

全実行時間＝スワップ時間＋実行時間

=アクセス時間+ロード時間+実行時間
$=10^{-2}+10^{-1}+1.0$
=1.11秒

8.7.2.2 デマンド・ページング・システム

本節でも，ジョブが開始されたら完了するまで実行するものと仮定する（$tq=\infty$）．図 8.11は，デマンド・ページングを行なうシステムにおいて例題プログラムを実行したときの時間に対する記憶域の利用率を示したものである．

図 8.11 では簡単のために，プログラムが5ミリ秒ごとに新しいページを参照するものとし（どのページも500ミリ秒経つと少なくとも1回はアクセスされる），デマンド・ページングを行なっていると仮定する．システムは例題プログラムの第1ページをアクセスしてロードする．このプログラムは5ミリ秒実行され次のページのアクセスとロードとが行なわれる．この過程はすべてのページがアクセスされるまで続けられる．その後，例題プログラムは完了するまで実行する．ページ読込みの間には計算を行なわないと仮定しておく．

全実行時間は次のように計算される：

全実行時間=(ページ数)×(ページ・スワップ時間)+実行時間
=100×(アクセス時間+ページ・ロード時間)+実行時間

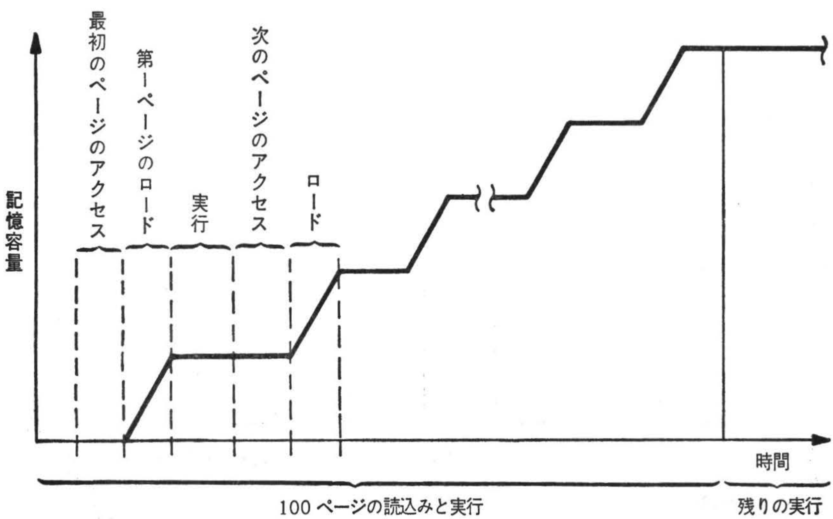


図 8.11 番号1でデマンド・ページングを行なう場合の使用記憶容量

$$=10^2\times(10^{-2}+10^{-3})+1.0$$
$$=2.1秒$$

8.7.3　場合2：割当て時間＝500ミリ秒

割当て時間を500ミリ秒として解析を繰返す．システムは500ミリ秒だけ1つのジョブを実行して，このジョブを主記憶から追出す．ジョブが500ミリ秒で終了しないならば，このジョブはシステムから追出され，後刻再ロードされ500ミリ秒の間実行される．この過程はジョブが完了するまで繰返される．

8.7.3.1　スワップ・システム

図8.12は，tq＝500ミリ秒の場合のスワップ・システムで例題プログラムを実行したときの記憶域利用率を示している．このような場合，tq＝500ミリ秒としてスワッピングを行なった場合には，2回のスワップ入力と1回のスワップ出力とが必要である．本節では，これらの時間は等しいとしておく．

全実行時間＝3×(スワップ時間)＋実行時間
$$=3\times0.11+1$$
$$=1.33秒$$

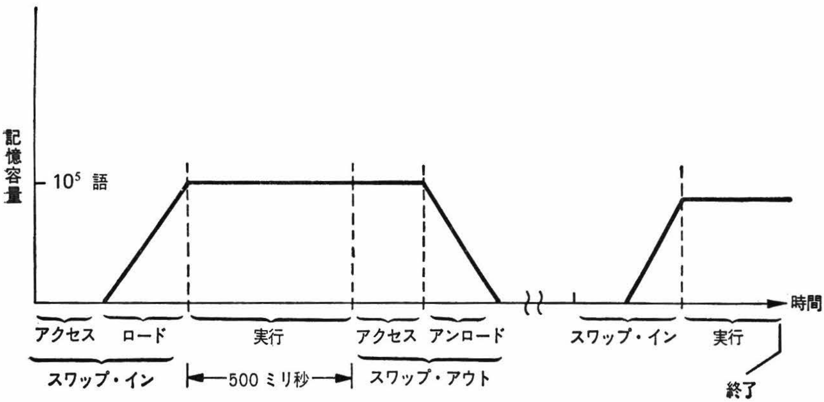


図8.12　番号2でスワッピングするシステム

8.7.3.2 デマンド・ページング・システム

前に述べたのと同様にページをロードするが，割当て時間が経過したらスワップ・システムと同様に，一度にすべてのページを追出す．このようにしてスワップすると記憶域を他の利用者のために空けることができる．この技法は多くのシステムで用いられており，**デマンドによってロードしポスト・ページを行なう方式**といわれる．

図 8.13 は，この場合の動特性を示している．

全実行時間は次のようにして計算される：

$$
\begin{aligned}
\text{全実行時間} &= 2\times 100\text{ページの取出し}+1\text{スワップ出力}+\text{実行}\\
&=2[100\times(10^{-2}+10^{-3})]+[10^{-2}+10^{-1}]+1\\
&=3.31\text{秒}
\end{aligned}
$$

8.7.4 場合 3：割当て時間＝50ミリ秒

割当て時間を50ミリ秒にしたらどうなるであろうか．このようにするとはるかに応答性のよいシステムができるであろう．

8.7.4.1 スワップ・システム

この場合には20回の読込みと19回の書出しが必要である：

$$
\begin{aligned}
\text{全実行時間} &= 39\times(\text{アクセス}+\text{ロード})+\text{実行}\\
&=39\times(10^{-2}+10^{-1})+1\\
&=5.29\text{秒}
\end{aligned}
$$

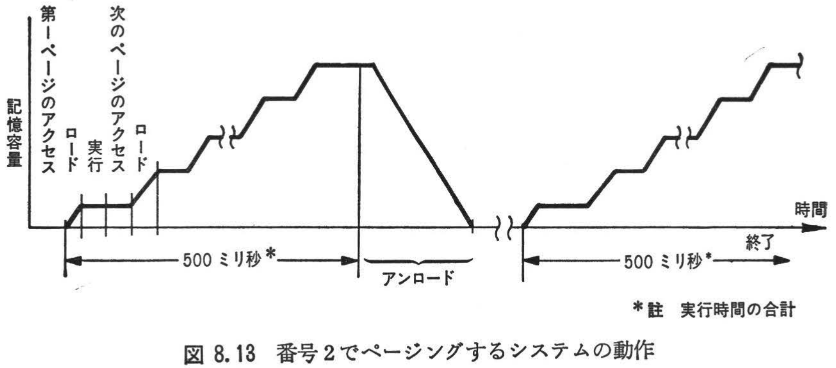


図 8.13　番号 2 でページングするシステムの動作

8.7.4.2　デマンド・ページング・システム

50ミリ秒の間には10ページだけがアクセスされ，毎回ロードとスワップが行なわれる：

$$\begin{aligned}\text{全実行時間} &= 20\times(10\text{ページの読込み})+19\text{スワップ}+\text{実行}\\ &= 20\times10\times(10^{-2}+10^{-3})+19\times(10^{-2}+10^{-2})+1\\ &= 3.58\text{秒}\end{aligned}$$

8.7.5　結　　論

図 8.14 は結果の要約を示している．タイムシェアリング・システムでは，割当て時間がかなり長い場合でも，入出力要求のために実行時間は 500 ミリ秒よりもはるかに短かくなる傾向がある．

システムの応答性に関連する *tq* の選択によって，スワッピングにするかデマンド・ページングにするかが決定される．*tq*=∞はバッチ・オペレーティング・システムに対応し，*tq*=500 ミリ秒や *tq*=50ミリ秒は会話型のタイムシェアリング・システムに対応する．

したがって，応答性がよくて仕事量の大きいシステムが必要ならば，ページング・システムを用いるのがよい．この簡単な例では，このジョブ混合体に対しては，バッチ・システムとしてスワッピング・システムが最適であることが

番　号	応　答	割当て時間	記憶管理方式	合計の実行時間
1	低　速	割当て時間=∞	スワッピング	1.11
			ページング	2.1
2	中　速	割当て時間=500ミリ秒	スワッピング	1.33
			ページング	3.31
3	高　速	割当て時間=50ミリ秒	スワッピング	5.29
			ページング	3.58

図 8.14　ページング対スワッピングの解析結果

わかる．

8.8 スラッシング

前節では，ページングが優れていることをみた．また，多重プログラミングの利点をも論じた．これら2つの技法を組合せると，利点ばかりを得ることができるであろうか．ページングは記憶管理の機能であり，多重プログラミングは処理装置管理の機能である．これらの2つは相互に関係するであろうか．これらは関係がある．その証拠に，これらを組合せるとシステム効率が著しく低下することがある．この性能低下は漠然と**スラッシング**と呼ばれている (Swartz, 1972 ; Denning, 1968 ; Randel, 1968)．スラッシングの定義には次に示すものがある：

1. スラッシングとは，システムがディスクを最高速度で駆動して多量の入出力を実行し，ディスクが MAYTAG† の洗濯機が乾燥のサイクルになっているような状況をいう．
2. スラッシングとは，システムが大量の仕事を行なっているが，結果を生じないような状態である．
3. スラッシングとは，多重プログラミングにおいて1つのジョブのために使用される CPU 時間の合計が，単一プログラム実行のときにこのジョブのために使用される CPU 時間の合計に等しい点をいう．すなわち，多重プログラミングが単一プログラミングより性能が悪くなる点をいう．

スラッシングを取扱う1つの方法は，これに関連した特定の問題を論じることである．本書では例を用いてこれを論じよう（スラッシング解析の別の手法については Denning (1968) を参照せよ)．

本節では，デマンド・ページングを行なうシステムにおいて，1つのジョブで使用される CPU 時間の合計（実行時間，入出力待合せ時間，ページング時間など）と多重プログラミングされるジョブ数との関係の形式的な解析を行なう．ここでは，1つのジョブのために使用される合計時間が単一プログラムでそのジョブに対して使用される合計時間と等しくなるような多重度が存在する

---

† MAYTAG 電気洗濯機の商品名： 洗濯物がドラムと一緒に回転して途中まで くると落下するさまを，ディスクのレコードが主記憶に入っても充分利用されない内にたえず出し入れされる状況にたとえている．

ことを示そう．この点が**スラッシング点**と呼ばれ，多重プログラミングが自滅する点である．

本書では，以下に示すような等しいジョブを幾つも実行して，多重プログラミング・システムでこの点がどこにあるかを示す．

8.8.1　例題ジョブ

8.8.1.1　計算機の特性

ハードウェアは IBM 360/370 に類似しており，次の特性を持っていると仮定する：

記憶容量＝256Kバイト

CPU 速度＝2×10^{-6}秒/1命令当り．

ドラムは1回転40ミリ秒で，平均アクセス20ミリ秒，転送時間はアクセス時間に比べて小さいとする．

他の入出力装置が存在する．ディスクや印刷機など．

MULTICS (Corbató, 1969) や CP-67 (Bard, 1973) と同様の“1ビット”の LRU 置換えアルゴリズムを用いたデマンド・ページング・システムを用いている．

4K バイトのページを用いる（ドラム・アクセスは20ミリ秒であり，1ページを取出すのに20ミリ秒かかる）．

8.8.1.2　プログラムの特性

本節は，例題プログラム（実際には IBM 360のアセンブラ）の特性とこれらを測定する方法について論じる．仕事量や効率を改善するためにこれらの特性を利用するのはオペレーティング・システムの機能である．システムにはこのプログラムだけが入っていると仮定する．すなわち，すべてのジョブはこのプログラムの写しを実行している．

プログラムには2種の特性が存在する．それらは，測定が簡単な外部特性と，測定するのが困難な内部特性である (Coffman, 1968；Brawn, 1968；Hatfield, 1972)．

外部特性

プログラムの大きさ＝160,000バイト

1,000,000命令の実行＝CPU　2秒

60回の入出力要求を出す．各入出力には50ミリ秒かかる．（したがって，多重プログラミングを行なわずに実行すると，入出力待合せ時間（$T_{I/O}$）＝3秒である．

したがって，入出力待合せ率＝60%である。

内部特性

第3.6.3節では，デマンド・ページングの場合のプログラム特性を表わす種々の方法を示した（図3.36にはプログラムの番地参照のパターンが示されている）．プログラムのページング特性を示す総合的な尺度は，記憶容量に対してページ割出しの回数を示した曲線（第3.6.2節参照）である．図8.15には，考察中の例題プログラムに対する主記憶容量とページ割込み回数の関係を示している．

8.8.2　例題プログラムを実行するために使用される合計のCPU時間の解析

本節の目的は，このジョブに対して使用される合計のCPU時間 T_{TOTAL}，すなわち，ジョブを実行し，ジョブの入出力を待合せ，ページ入出力を待合せ，ページ入出力を待合せるために使用されたCPU時間を計算することである．この時間を，多重プログラミングするジョブ数の関数として求めよう．すなわち，1ジョブのときの T_{TOTAL}，2ジョブのときの T_{TOTAL}，・・・を求めよう．記憶装置は各ジョブに対して均等に分割されていると仮定する（たとえば，システムに4つのジョブが存在していると，各ジョブには64Kバイトが与えられる）．このようなとき，多くのジョブがシステムに入ると，各プログラムで使用できる記憶域が小さくなるのでページ呼出しの回数が増大する（図8.15）．

8.8.2.1　課金の方式

T_{TOTAL}, T_{CPU}, $T_{I/O}$, および T_{PAGE} を次のように定義しよう：

T_{CPU}＝実行のためのCPU時間

$T_{I/O}$＝同時実行していない入出力時間

T_{PAGE}＝同時実行していないページ入出力時間

上の項目を定義するについては，すべてのCPU時間はいずれかの利用者に課金されるものと仮定した課金方針を用いている．言葉を換えると，オーバヘ

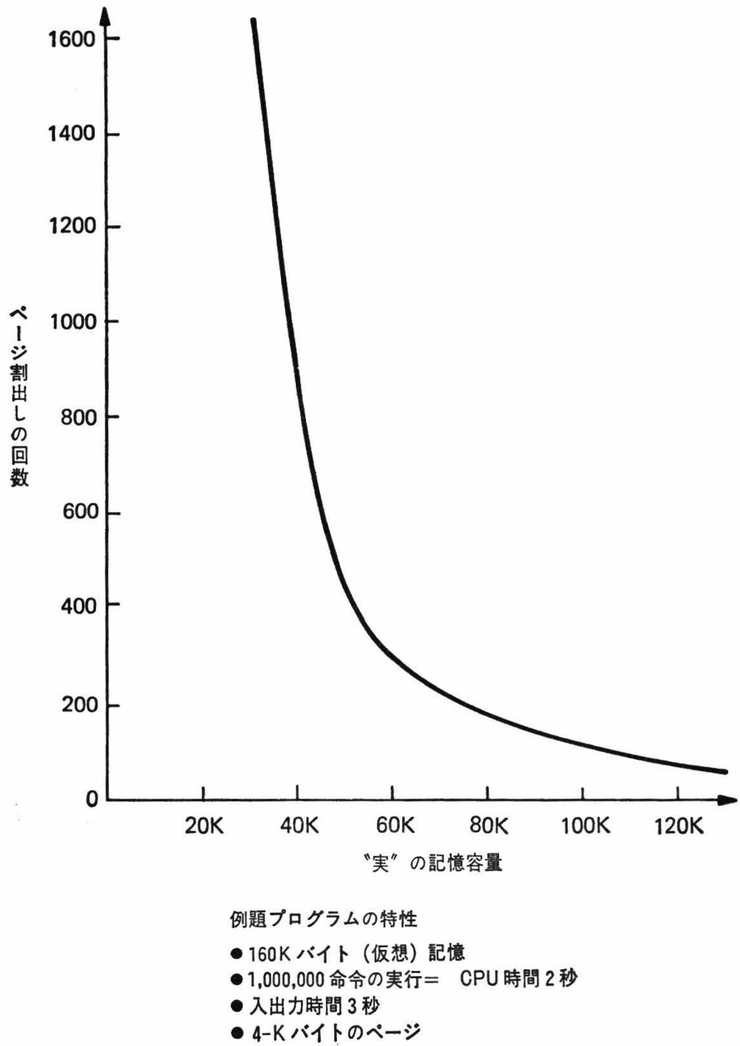


図 8.15　例題プログラムのページング動作の実験結果

ッド分というものはなく，CPU を使用すればその分の料金を支払うことになる．

ジョブ入出力要求を出し，この時点では他のジョブが実行できないならば，アイドルになった CPU 時間 $T_{I/O}$ の分を支払わねばならない．ジョブがペー

ジ割出しを起こしたとき，ページを処理している間にCPUが実行できるようなジョブが存在しないならば，CPUのアイドル時間 T_{PAGE} を支払わねばならない.

このような方式は，入出力を実行したときCPUがほかに仕事を持たなかったのは運が悪いのであって，CPUが他のジョブを実行できるのは運がよいだけであるから，不公平であると考えるかもしれない．しかし，このようなことは，長期的には平均化される.

システムの中のジョブ数の関数として T_{TOTAL} を求めたい．そこで，T_{CPU}, $T_{I/O}$, および T_{PAGE} をシステムで多重プログラミングされているジョブ数の関数として計算してみよう．この例ではすべてのジョブは同一のジョブを実行する．図8.16は T_{CPU}, $T_{I/O}$, および T_{PAGE} の値を示している．これらの値は次のようにして計算される.

8.8.2.2 T_{CPU}

T_{CPU} は，ジョブの命令を実行するためにCPUが使用する時間である．この値は，システムに幾つジョブが入っていても，一定である.

	入出力		ページング					CPU	合計	利用率
ジョブ数	同時実行しない入出力待合せ率 %	$T_{I/O}$	1ジョブ当たりの記憶容量	1ジョブ当たりのページ割出しの回数	1ジョブ当たりのページ待合せ率%	同時実行しないページ待合せ率%	T_{PAGE}	T_{CPU}	T_{TOTAL}	%
1	60.0	3.00	256	25	20.0	20.0	.500	2.000	5.500	36.36
2	31.0	.898	128	33	24.8	4.0	.084	2.000	2.982	67.07
3	13.4	.310	85	185	64.9	18.9	.466	2.000	2.776	72.05
4	04.8	.100	64	290	74.4	20.6	.518	2.000	2.618	76.39
5	01.4	.028	51	415	80.6	19.9	.496	2.000	2.524	79.24
6	00.4	.008	43	600	85.7	24.1	.636	2.000	2.644	75.64
7	00.1	.002	37	1200	92.3	36.2	1.134	2.000	3.136	63.78
8	00.01	.0002	32	1600	94.1	60.9	3.116	2.000	5.116	39.09
1	2	3	4	5	6	7	8	9	10	11

図 8.16 $T_{I/O}$, T_{PAGE}, T_{CPU} の結果

$$T_{CPU}=(10^6\ 命令)\times(2\times10^{-6}秒／命令)=2秒$$

8.8.2.3　$T_{I/O}$

1つのジョブは，入出力要求を出す前に短時間実行する．単一ジョブの場合の $T_{I/O}$（入出力待合せ）は3秒である．これは例題プログラムの特性の1つであった。

2つのジョブが実行している場合には，第1のジョブが入出力を待合せている間 CPU は第2のジョブを実行することができるので，入出力待合せ時間は小さくなる．第3.2.2節の解析法と表3.1を用いて，$T_{I/O}$ の欄をすべて記入することができる．

w を入出力が同時実行されない確率とすると次の関係が求められる：

$$T_{I/O}=w\times(実行時間)$$
$$=w\times(T_{CPU}+T_{I/O})$$

したがって，

$$T_{I/O}=\frac{w}{1-w}\times T_{CPU}$$

表の第3欄は，第2欄と T_{CPU} から計算することができる．

8.8.2.4　T_{PAGE}

T_{PAGE} を計算するには，1つのページを待合せて CPU がどれだけの期間アイドルになるかを知る必要がある．T_{PAGE} の中間的な段階は図8.16の第4, 5, 6, 7，および8欄に示されている．

まず，各ジョブは，ジョブ数で主記憶容量を除した値に等しい記憶域を与えられていると仮定する．この値が“ジョブごとの記憶域”の欄に示されている．ジョブごとの記憶域の値を用いて，図8.15から“ジョブごとのページ割出しの回数”欄が作られる．ジョブが単独で実行していると仮定して，“ジョブごとの待合せ率”欄が計算される．この計算式は次のとおりである：

$$\frac{(ページ割出しの回数)\times(1つのページ割出しの処理時間)}{(ページ割出しの回数)\times(1つのページ割出しの処理時間)+T_{CPU}}$$

この欄は，ページ割出し処理時間を20ミリ秒，$T_{CPU}=2$ 秒と仮定して計算された．次に，同時実行されない入出力待合せ時間の場合と同じマルコフのモデル

と表3.1とを用いて，単一ジョブの場合のページ待合せの値から“同時実行されないページ待合せ率”を計算することができる．この待合せ率を w とすると，T_{PAGE} を計算するのに次式を用いることができる：

$$T_{\mathrm{PAGE}}=\frac{w'}{1-w'}\times T_{\mathrm{CPU}}$$ （$T_{\mathrm{I/O}}$と同様の導出による）

したがって，

$$T_{\mathrm{TOTAL}}=T_{\mathrm{CPU}}+T_{\mathrm{I/O}}+T_{\mathrm{PAGE}}$$

8.8.2.5 T_{TOTAL}

図 8.17 には結果が要約されており，図8.16には曲線として示されている．]

8.8.3 結　　論

8.8.3.1 最適ジョブ数

図 8.18 から，最適ジョブ数は3〜6の範囲にあることがわかる．9つ以上のジョブを同時に実行すると，各ジョブを単一ジョブで実行するより多重プログラムで実行する方がCPU時間が多く必要となる．例として，15人の利用者を多重プログラミングで実行しようとすると，各ジョブには約16Kバイト（4ページ）が与えられる．T_{TOTAL} は32倍（ジョブごとに160秒以上）で3,200%にも増大する．CPU 時間の内の 5〜10%だけが T_{CPU} として使用されているに過ぎず，スラッシングの極端な場合である．

ジョブ数	T_{CPU}	$T_{\mathrm{I/O}}$	T_{PAGE}	T_{TOTAL}	CPU 利用率%
1	2.000	3.000	.500	5.500	36.36
2	2.000	.898	.084	2.982	67.07
3	2.000	.310	.466	2.776	72.05
4	2.000	.100	.518	2.618	76.39
5	2.000	.028	.496	2.524	79.24
6	2.000	.008	.636	2.644	75.64
7	2.000	.002	1.134	3.136	63.78
8	2.000	.0002	3.116	5.116	39.09

図 8.17 総合的な性能： 入出力とページングを独立に取扱っている

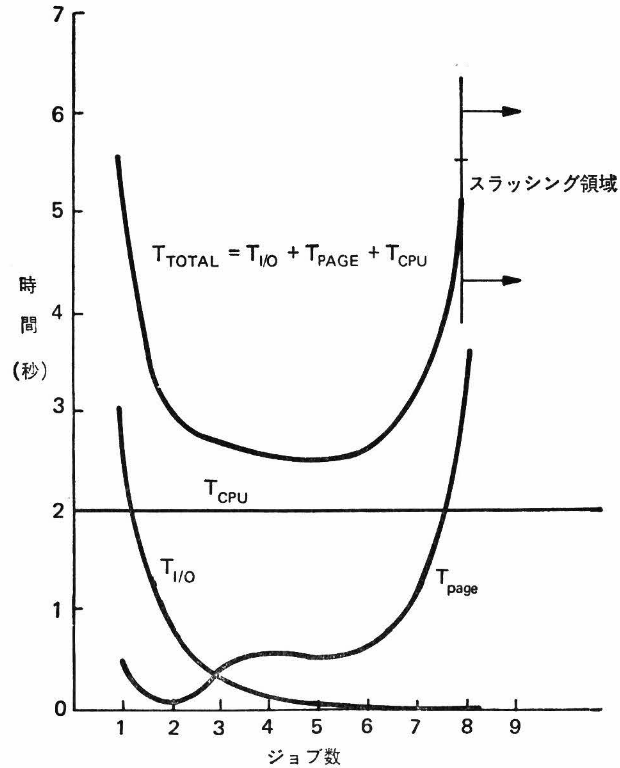


図 8.18　例題プログラムの解析：　同時実行しない処理時間

システム設計者の仕事の1つは，多重プログラミングのジョブ数を決定することである．実際には，ジョブの特性は変化するので，ジョブ数の範囲を決定する．普通この範囲は，たいていのシステムでは4〜8ジョブの範囲にある．第9章で述べる MULTICS や VM/370 システムではスラッシングを避けるために多重プログラミングのジョブ数を制限している．読者はこれらのシステムの説明を参照してほしい．

8.8.3.2　仮　　定

スラッシングの解析では，簡単のために幾つかの仮定を行なった．もっと複雑な解析も行なわれているが（Buzen, 1973；Sekino, 1972），その結果は基本

的に図 8.17 に示したものと同じである．しかし，3つの仮定をもう少し吟味しておこう：

1. **$T_{I/O}$ と T_{PAGE} とは独立である**　前述の解析では $T_{I/O}$ と T_{PAGE} とは独立であると仮定した．しかし，実際には，ジョブが入出力を待合せているときにはページ割出しを起こすことはなく，その逆も言えるので，これらは独立ではない．これらの関係を調べるために行列のモデルが作られている（Schwartz, 1973）．その結果は，前に求めた結果と対比して図 8.19 と図 8.20 に示されている．単純な解析のものと個々の値は異なっているが，曲線の全体的な形やスラッシング領域の点については全くよく似ている．
2. **各ジョブは等容量の記憶域を受取る**　図 8.19 において実験的に観測され報告されたページ割出しの関係を用いて，各ジョブに等しく記憶域を分割したのは実はよい方策であったことを示そう．すなわち，この記憶域の分割法は，図 8.18 に示したスラッシングの状態を悪化させることにはならない．

記憶域割付けで4つのジョブがあるとし，各利用者に64Kバイトが与えられるとしよう．このとき，図 8.15 を用いて，システム全体のページ割出しは次のようになる：

$$4\times293=1,172 \quad \text{ページ割出し}$$

ここで，8Kの記憶域を1人の利用者から他の利用者に移してみよう（たとえば，2人の利用者は64K，1人は56K，そして4人目は72Kの区画を持つ）．

ジョブ数	CPU 利用率 %
1	36.4
2	62.8
3	70.3
4	65.2
5	66.3
6	63.5
7	47.1
8	43.4

図 8.19　総合的な性能：入出力待合せとページ待合せのモデルを複合

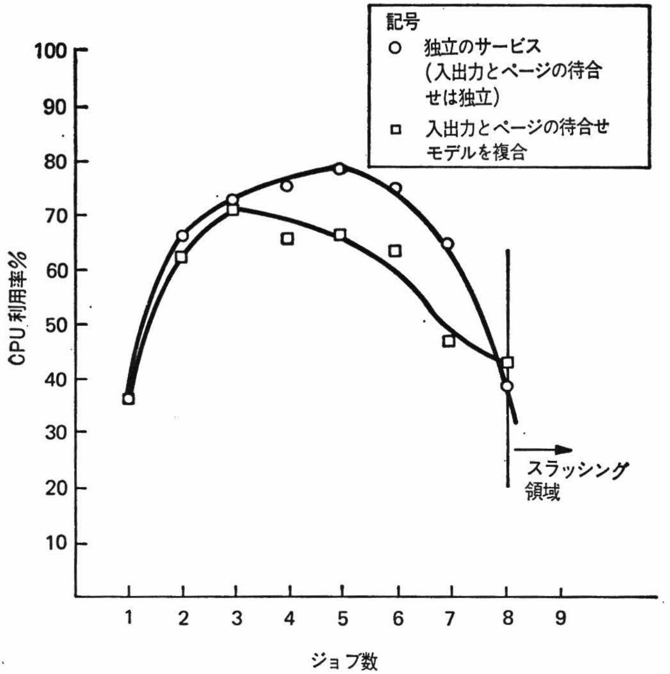


図 8.20　例題プログラムの解析：CPU 利用率

$$合計のページ割出し回数=2\times293+359+237=1,182$$

これはページ割出しの回数が10回増加している．主記憶容量とページ割込み回数の間係を用いる複雑さと単純なラウンド・ロビンのスケジューリングを考えてみて，各ジョブに等しい容量の領域を与えることが合理的である．しかし，この方法は別のスケジューリング・アルゴリズムのときには必ずしも最良の方法ではない．

3.　入出力およびページングの時間が一定である　本書の解析では入出力とページ要求に対するサービス時間は一定であると仮定した．実際のシステムでは限られた台数のチャネルや装置しか存在しないので，チャネルや入出力装置の容量を越えた負荷によって生じる行列処理の遅れのために図8.18に示したよりもシステム性能が低下する．

8.8.3.3 スラッシングに影響する要因

本書の解析はただ1つの場合を論じたに過ぎない．特定のシステムを解析する場合に考慮すべきその他の要因には次のものが考えられる：

プログラムの性質（ページ割出しの特性，入出力量，CPU 使用量）

ハードウェアの種類（記憶容量，ドラム速度など）

オペレーティング・システムの種類（記憶装置および処理装置の管理アルゴリズム）

8.9 結　論

“最良”の記憶装置，処理装置，装置，あるいは情報管理方式を用いて，これらを1つのシステムとして組合せたとしても，必ずしも“最良”のシステムができるとは限らない．

本書では，システム設計者や利用者が自分に合うようにシステムを改善するための2,3の相互関係について触れた．これらの関係を測定して，システム決定を行なうために必要な情報を求める解析技法について説明した．このほかに多くの解析例や技法が文献表に示された参考資料に示されている．

演 習 問 題

8.1 第8.1節は，各区割が500,000バイトであると仮定して，記憶管理の解析を行なっている．区割の容量が100,000バイトである場合には解析結果はどのようになるか．

8.2 スケジューリングは，資源管理に含まれる基本的な処理の1つである．基本的には，複数のプロセス間で共用する必要のある資源は限定されている．システムがどの程度うまく作られているかは，資源をスケジュールするアルゴリズムがどの程度うまく働いているかによって測定することができる．主要な4つの資源（記憶装置，処理装置，装置，情報）の内の2つについて次の問に答えよ：

1. これらの資源をスケジュールするアルゴリズムがどの程度うまく働いているかを測定するための一般的な方法を示せ．その方法が優れている理由を説明せよ．

2. 2つの資源のためのスケジューリング・アルゴリズムで生じる相互の関係について論じよ．
3. 問1の答えから作成された2つの資源のためのスケジューリング・アルゴリズムの類似している点を示せ．言葉を換えると，一般に資源をスケジュールするときに存在する類似点は何か．相異点は何か．

8.3

1. 再配置型の区画が固定の区画より優れているようなジョブの集合と特性の例を示せ．仮定した記憶装置と CPU の速度を示せ．
2. 同じジョブ集合を用いて，固定の区画が再配置型の区画より優れているような CPU と記憶装置の速度を示せ．

8.4 本問の目的は，図 8.3に示した現象，すなわち，多重プログラミングのジョブ数が多過ぎるとかえって性能が悪くなることを示すような計算を行なうことである．次の仮定を行なう：

1. ページングを行なわない多重プログラミング・システムがある．
2. システムには16台のディスク装置がある．
3. ジョブは順編成の入出力を行なう．すなわち，プログラムは同一のシリンダか隣接するシリンダをアクセスする傾向がある．
4. 各ジョブは2つのディスク・ファイルを持っている．
5. 同一または隣接シリンダのファイルに対する入出力アクセス時間は20ミリ秒である．これ以外のファイルへの入出力アクセス時間は50ミリ秒である．
6. ジョブは平均20回の入出力を行なう．単一プログラム実行の場合には入出力待合せは40%である．しかし，本問の多重プログラミング・システムでは，多重プログラミングのジョブ数を増してある点を越えると次の2つの要因によってジョブの入出力待合せ率は増大する．
   a．ジョブは，他のジョブが使用している装置を待合せることが必要となる．たとえば，システムにジョブが2つしか存在しないとしたら，4つのファイルは同一のディスクには置かれないであろう．しかし，システムに64個のジョブが入っているときには，128個のファイルで同一のディスクに置かれるものができる．
   b．単一プログラム実行の場合には，たいていのディスクのアクセスはディスクのシリンダを順次参照するであろう．しかし，本問の例では，ジョブ数が増大すると，ディスクの腕が正しい位置にない確率が

大きくなる.

このような理由によって，多重度がある値を越すと各ジョブの実効的な入出力の速度が低下する.

1. ジョブの使用する CPU 時間を計算せよ.
2. ディスクAのファイルを2回アクセスする間に他のジョブがディスクAをアクセスする確立を計算せよ．この値を用いて，多重プログラミングの場合の入出力アクセス時間の期待値を，ジョブ数Nの関数として求めよ.
3. ディスクAのファイルを2回アクセスする間にディスクAに対して行なわれる入出力要求の期待値を計算せよ.
4. 2回の入出力動作の間にジョブが使用する CPU 時間の平均値は幾らか.
5. 統計的にみて，このシステムでは1つのディスクに要求が溜まることがあるか.
6. ジョブの入出力待合せの期待値をジョブ数の関数として求めよ．また，入出力待合せ率の期待値をジョブ数の関数として求めよ.
7. 上記の値を最大20ジョブまで計算せよ.

8.5 システムでは多重プログラミングが行なわれていて主記憶に4つのジョブが入っているとする．各ジョブの入出力待合せの平均値は20%であるとすると，上手な装置管理技法を用いて節約する CPU 時間の最大値はどれだけか．読者の答えを説明せよ.

8.6 IBM の計算機販売員が読者の会社に新しい“改良型”の装置を売込みにやってきた（価格はわずか高い程度である）．読者は，提案されたシステム変更が CPU の利用率にどう影響するかを評価する必要がある.

次に述べるような構成が考えられる：

1. 読者のプログラムでの入出力はほとんど印刷出力（スプーリングされている）であるので，印刷機を1,000行/分から10,000行/分にし，スプール用ディスクを平均アクセス時間 200 ミリ秒のものから平均アクセス時間20ミリ秒のものに変更して能率向上を計ることができる.
2. ページング・ドラムを平均アクセス時間20ミリ秒のものから平均アクセス時間2ミリ秒以下のものに変更することができる.
3. 記憶容量を256Kバイトから200万バイトに変更することができる.

4. CPU と記憶装置とを変更して，現在の500K命令／秒から 200 万命令／秒のシステム 380 に変更することができる.
5. 上記のすべてを実行することもできる.

上述のシステム変更については次のことを行なわなければならない：

1. どの T（図 8.18 で示した T_{CPU}, $T_{I/O}$, T_{PAGE}）が影響を受けるか示せ．どのような影響を受けるか．その理由は何か.
2. 実行すべき最適ジョブ数にどのように影響するか述べよ(図8.20)．すなわち，最大値を与える点は左へ動くか右へ動くか．その理由は何か.
3. 図 8.20を基本として，システム変更1～5における CPU 利用率の曲線を描け.
4. これらの変更で支払うべき金額について簡単に論じよ.

8.7 第 8.7 節で行なったページングとスワッピングの比較の解析をもっと現実的なモデルを加えて行なえ．すなわち，主記憶中にある程度のページが入るまではページ割出しの発生は多い．この点を越えるとページ割出しの回数は減少する．プログラムのページングの動作特性を規定するために図 8.15を使用せよ.

第 9 章

事例研究

9.1 はじめに

本章の目的は次の 4 つである：

1. 実際のオペレーティング・システムで使用されている重要なアルゴリズムを説明すること.
2. 実際のシステムを説明して，本書の考え方の適応性を示すこと.
3. 取上げたシステムを研究する糸口を与えること.
4. 次のシステムを述べること.
   a. IBM の 370 VS1, VS2；360 MVT, MFT, BOS
   これらは広く使用されているので選ばれた.
   b. コンパチブル・タイムシェアリング・システム (CTSS)
   これは初期のタイムシェアリング・システムとして歴史的に重要であり，幾つかの優れた技法を含んでいる.
   c. MULTICS
   進んだ特徴と概念を持つタイムシェアリング・システムである.
   d. VM 370 と CP-67/CMS
   仮想計算機の概念の例である.

選択したシステムは，バッチ，タイムシェアリング，および実時間システムのよい例である．すべてのオペレーティング・システムは同じ機能を果たす．すなわち，機構は同一であってただ方針が異なっているのである．その例は次のようなものである：

バッチ・システムのスケジューリングの方針は平均的な仕事量を最大にすることである.

タイムシェアリング・システムのスケジューリングの方針は応答性を高めることである．応答時間は利用者の期待にそうものでなくてはならない．利用者は，仕事がやさしいときには計算機が迅速に仕事をすることを要求する．したがって，タイムシェアリング・システムは会話型の仕事を行なう利用者や簡単な仕事を優先して処理する．

実時間システムのスケジューリングの方針は，システムが大量のデータを処理できるようにすることである．このシステムは，データが入ってくるよりも速くデータの解析と処理を行なう必要があり，時間的な事象にも応答しなければならない．

これらのシステムはすべて資源管理を有しているが，管理の方針が異なっている．1つのよい演習として，第7章の例題オペレーティング・システムを基にして，次の問題を解いてみよ：

1. 優れたバッチ・システムとする．
2. 優れたタイムシェアリング・システムとする．
3. 優れた実時間システムとする．

本書の目的は各システムを詳細に述べようとすることではなく，それぞれの重要な特徴を示して適当な参考図書を探せるようにすることである．すべてのシステムを述べようともしていないし，主要なものを忘れずに述べようともしていない．読者がこれ以外のシステムを解析するときに本書の考え方を役立ててもらえれば幸いである．

実際のオペレーティング・システムの設計がつねに不変であることは滅多にないことに注意してほしい．オペレーティング・システムはつねに変更されるので，最新の情報をすべて包含するような記述は，不可能ではないにしても，実行するのが困難である．このような理由から，以下の事例研究は実際のシステムで見られるような資源管理の技法とアルゴリズムの例としてみてほしい．これらの例は，読者の経験と洞察力とを拡めるだけではなく，他のシステムを研究するための手法を与え，概説したシステムをより深く研究するための出発点ともなろう．

これらのシステムの幾つかの特徴は前章までに論じているので，説明がきわめて短いところもある．

9.2 IBM システム360と370のオペレーティング・システム

9.2.1 体　　系

IBM のシステム360とシステム370のためのオペレーティング・システムは，1964年以来発展してきた系統の一部である．図9.1は主要なオペレーティング・システムを示し，図9.2は，これらを幾らかは互換性のある2つの群に分けて示したものである．これらのシステムは多数の任意選択機能を持っている．本書では主として OS/360 の流れである第2群に焦点を合せよう．

これらのオペレーティング・システムの最も特徴的なものは記憶管理ルーチンである．PCP (Primary Control Program) は小規模な360システムのために作られた初期のオペレーティング・システムである．したがって，このシステムは単純な単一連続割付けの記憶管理方式を用いていた (IBM ではもう

標 準
BPS/360 (基本プログラミング・サポート)
BOS/360 (基本オペレーティング・システム)
TOS/360 (テープ・オペレーティング・システム)
* DOS/360 (ディスク・オペレーティング・システム)
* OS/360-PCP (基本制御プログラム)
* OS/360-MFT (固定タスク数の多重プログラミング)
* OS/360-MVT (可変タスク数の多重プログラミング)
* OS/360-VMS (可変記憶容量システム)
* CP/67
TSS/360 ― 360/67 のみ
DOS/VS
* OS/VS1
* OS/VS2
* OS/VS2 REL2
* VM/370
(DOS/VS ～ VM/370：仮想システム) 370 のみ
複雑さの順番

その他：(大部分は利用者が作成したもの，またはOS の拡張)
HASP
ASP
LASP
MTS

* = 本書で論じるシステム

図 9.1　システム360/370のオペレーティング・システム

群1	群2
BOS TOS DOS DOS - II (現在のDOS) DOS/VS	OS/360-PCP OS/360-MFT OS/360-MFT-II(現在のOS/360-MFT) OS/360-MVT OS/VS1 OS/VS2
小規模構成の360/370用 360/25, 360/30, 360/40 370/125, 370/135 など	中～大規模構成の360/370用 360/50, 360/65, 360/75, 360/85 370/145, 370/158, 370/168など
単 純	より複雑

註：群1の中および群2の中ではシステムには互換性がかなりある．
しかし，これら2つの群の互換性はあまりない(IBMおよび顧客にとって厄介な問題である)．

図 9.2　システム360/370オペレーティング・システムの主要な2つのグループ

PCP を提供していない)．MFT (固定タスク数の多重プログラム) は，中規模の360のために作られた．このシステムの核部分は約64Kバイト必要である．このシステムは固定の区画割付け記憶管理方式を用いている．MFT は，あらかじめジョブ混合体に区画容量を適合させることができれば，優れたシステムである．MVT (可変タスク数の多重プログラミング) は，大規模の360のために作られた容量が可変の区画を用いるシステムで，核部分は128Kバイト以上必要である (IBM,16)．このシステムは，記憶域の要求量が予測できないようなジョブ混合体のときには優れたシステムである．

OS/360 の VMS (可変記憶容量システム) は，利用者があらかじめジョブ容量を指定しなくてもよいようにしたシステムであるが，遂に発表されることはなかった．VMS は MVT より前に作られ，動的領域割付けを行なうソフトウェア的なセグメンテーションを行なった (再配置は行なわない)．これらの記憶管理技法については前に第3.3節で論じた．

OS/VS1 と OS/VS2 は，それぞれ MFT および MVT に基づいているが，370 のページング・ハードウェアを利用して，利用者に大容量の仮想記憶を与えている (IBM, 14；IBM, 15)．CP-67 と VM/370 は仮想計算機のモニタを持ち，第9.5節で論じる．

370のハードウェアは，融通性に富むオペレーティング・システムの構成を

可能とし，オペレーティング・システムが資源をより効率よく利用できるようにした (IBM, 5)．特に，370は動的番地変換機構 (DAT) を備えている (IBM, 17)．これは，第3.5.1節および第3.6.1節で論じたページングと簡単なセグメンテーションの機構から成っている．セグメンテーションは256セグメントに限定されているので，256Kセグメントを使用できる Honeywell 6180 のような一般的な使用はできない．VS1はセグメンテーションのハードウェアを使用せず，VS2 はごく一部だけでこれを使用している．

装置管理を援助するために，370 は，第5.5.3.1節で説明したような回転位置検出機能を持ったブロック・マルチプレクサ・チャネルをも備えている．

9.2.2 記憶管理

9.2.2.1 分割割付け記憶管理

第3章で論じたように，MFT は静的な区画割付けの記憶管理方式を用い，MVT は可変の区画割付け方式を用いている．MVT でも MFT でも，利用者は必要とする最大記憶容量をあらかじめ指定しなければならない．まぼろしのシステムである VMS ではこのような必要はなく，ジョブが必要とすると領域を動的に割付ける．例として，図9.3はジョブ1とジョブ2が領域を要求して，VMS が動的に領域を割付けた状況を示している．しかし，この結果深刻な断片化の問題が発生する．

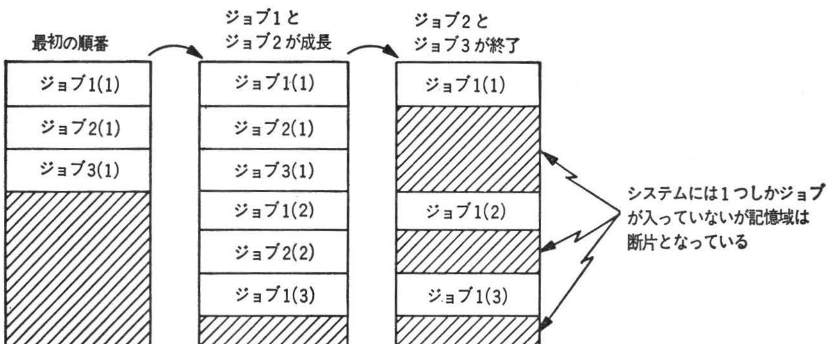


図 9.3 VMS の記憶割付け

9.2.2.2 デマンド・ページングの記憶管理

OS/VS1（仮想記憶1）はOS/360 MFTに基づき，OS/VS2（仮想記憶2）は OS/360 MVT に基づいている．両者は共に MFT および MVT と同じ区画割付けの方式をとっているが，これは実際の記憶域ではなくて仮想記憶での話である（図9.4参照）．

仮想記憶で MFT 型および MVT 型の方式を用いると次のような利点がある：

1. MFTおよびMVTをわずかに変更するだけでよい（これが主たる利点である）．
2. 記憶容量が大きい．
   a．断片の経費が小さくなる（仮想記憶上の断片は安価である）．
   b．スケジューリングが容易．
   c．経済的な“既成服”で間に合う（たとえば，大きな区画を要求しておいてごく一部を使用すること）．
3. 360/370の構成をそのまま伝える．
4. OSの任意選択機能の範囲を大きくできる（たとえば，スプーリング）――このようにしてもページングされた区画が利用できるので経費的には安上りである．

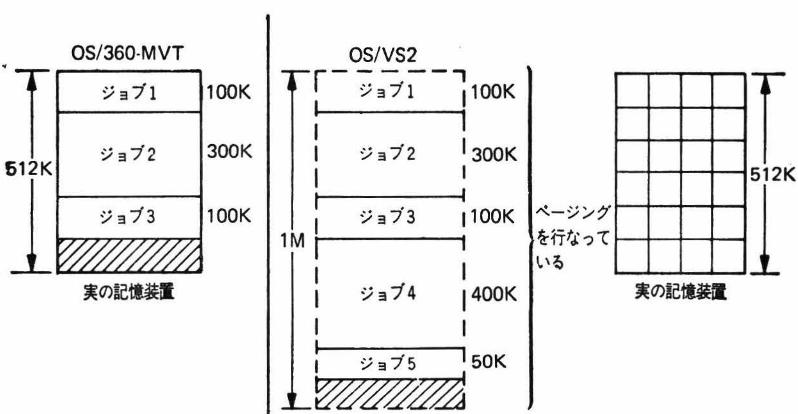


図 9.4 MVT および OS/VS2 の記憶管理方式

この方式の欠点は次のとおりである：

1. セグメンテーションを有効に使用していない.（VS2の区画は64Kのセグメントの整数倍である．記憶保護のためには，最大12区画という制限のつく記憶保護キーを用いるよりも，セグメント割出しのハードウェアを利用することが可能である.）
2. すべてのジョブの番地空間の合計は 16M 以下である.
3. MVT 以上の共有は不可能である（より大きい記憶域が利用できるので MVT 型の共有はずっと容易に行なえる).

他の問題点も存在する．たとえば，VS1 と VS2 の仮想記憶は，実際の記憶容量の 1.5〜2 倍程度の限られた容量で使用するために作られている．(16Mという制限もこの限度に関係する.）したがって，仮想記憶が実の記憶容量の 10〜100倍でページング（ページの入換え）が毎秒50〜100ページであるようなタイムシェアリング・システムと比べると，ページングの量は少ない（毎秒 5〜10ページ程度).

VS2の初版は第2版のときに大幅に変更されて，各ジョブの区画を最大16Mまでの独立の仮想記憶とすることができるようになった．図 9.5 は VS2 の第1版と第2版の仮想記憶の概念を示している.

VS1 と VS2 のページ置換えアルゴリズムは変形 **LRU** 方式である．1ビットの“参照ビット”を用いて，ページが参照されたかどうかを検査する．この

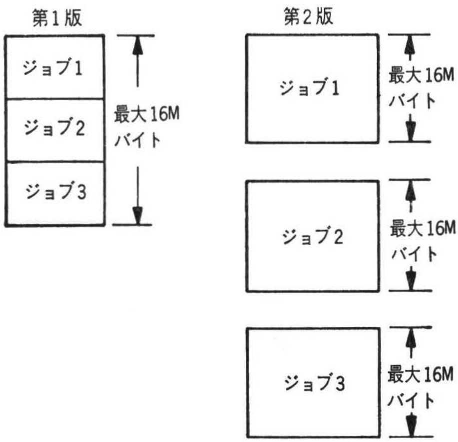


図 9.5 OS/VS2 第1版および第2版の仮想記憶

ビットを定期的に調べて図9.6のようなリストを作ると，VSは最も最近使用されていないページの順に並んだリストを作ることになる．図に示した集合の個数とビットの判定の回数が多いほど，ワーキング・セットは正確に管理される．VS1とVS2はページングの割合が小さいので，あまり多数のリストや多数回の判定を必要としない．

9.2.3　処理装置管理

9.2.3.1　ジョブ・スケジューラ

360/370のオペレーティング・システムの大部分は類似のジョブ・スケジューラを持っている．基本的には，これらはすべて優先順位によるスケジューラであり，大きな融通性を有している．例として，MFTを考えてみよう．次の4点が重要である：

1. 操作者は，各区画で処理することができるジョブ種別を設定する．各区画は，図9.7に示したように，プロセス・スケジューラの優先順位が定められている．
2. すべてのジョブはA, B, C, ・・・, X, Y, Zジョブに分類される．各クラスに入る条件は利用者によって定められるか，操作者によって指

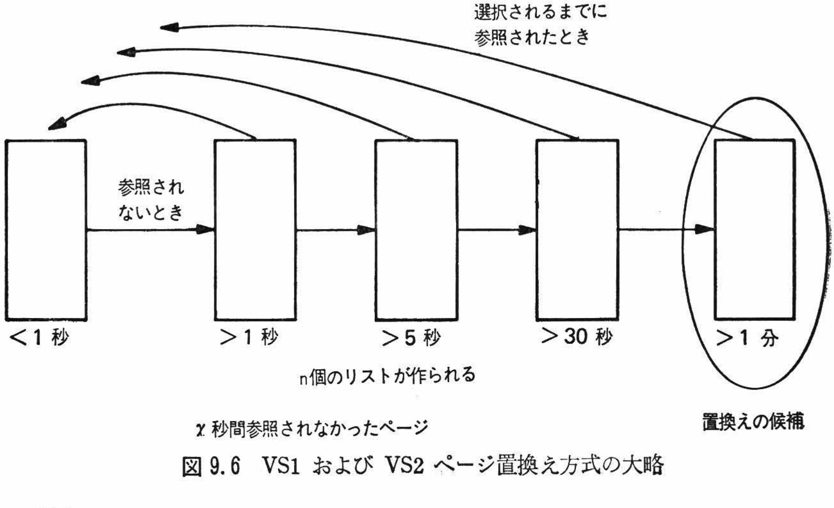


図9.6　VS1およびVS2ページ置換え方式の大略

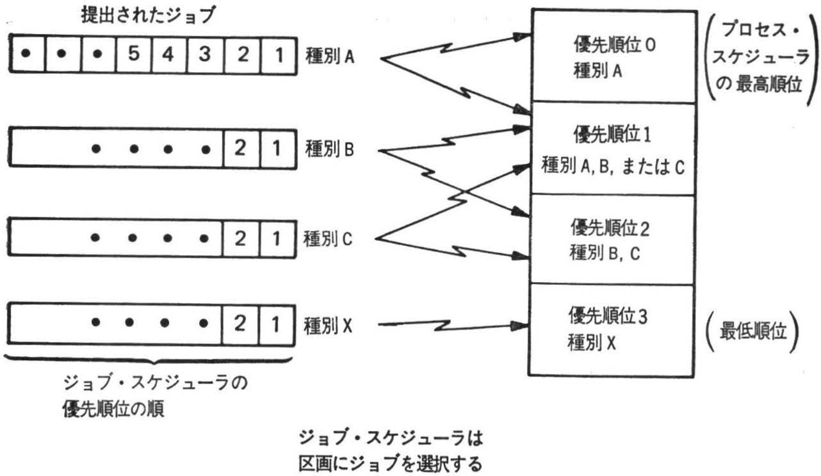


図 9.7 MFTシステムにおけるジョブ種別割付けの例

定される．典型的な方法は次のとおりである：

種別A――入出力限界のジョブ
種別B――平衡したジョブ
種別C――CPU 限界のジョブ
種別X――急行

3. 各利用者はジョブ・カードにジョブ種別とその種別の中での優先順位を指定する．
4. 操作者は，ジョブ種別を偽って申告しないような課金の方針を設定する．たとえば，種別 A のジョブは，入出力の経費は 10％の割引きがあり，CPU については15％の割増しとなる．この反対に，種別 C のジョブは， CPU の料金に割引きがある．このようにすると，種別Cのジョブを A として提出すると料金が高くなる．

基本的には，ジョブ・スケジューラまたはスプーラは提出されたジョブを種別ごとに並べ，次いで同じ種別の中で優先順位の順に並べる．1つの区画は2種別以上のジョブを処理することもできる．この場合には，種別は1次，2次というように順位づけられる．MFT で1つの区画が解放されると，スケジューラはその区画で処理される最も順位の高い種別の中の最も優先順位の高いジ

ョブを選択する．

図9.7の構成（種別Aは入出力限界，種別Bは平衡，種別CはCPU限界）では，ジョブ・スケジューリングでもプロセス・スケジューリングでも入出力限界のジョブの方が先に処理される．種別Cのジョブを持つ利用者は，種別Aの方が早く実行されるので，ジョブを種別Aと偽るかもしれない．しかし，上述の料金を課すことにすると，この利用者の料金は高くなる．もちろんオペレータはある種別のジョブを優先して処理するようにシステムを構成することもできる．

MVTでは，区画の個数が1から15まで変化するので，オペレータは区画に対して優先順位や種別をどのようにして設定すればよいであろうか．MVTでは，各区画に対して**イニシエータ**と呼ばれるプログラムを備えている．オペレータは，記憶域を求めて区画を作ろうと“競合”するジョブの種別と優先順位に関する情報を各イニシエータに与える．

MVTとVS2には，特定のジョブ混合体の要求に合うように，システム編集時にジョブ・スケジューラを調整する選択機能が備わっている．1つの例はロール・アウトである．高順位のジョブがシステムに入ると，そのジョブ種別の区画の中で実行している低順位ジョブはロール・アウトされる．

9.2.3.2 プロセス・スケジューラ

プロセス・スケジューラは優先順位に基づいて処理を行なっている．このスケジューラは，つねに，最高順位の区画で実行しているジョブに処理装置を割当てる．プロセス・スケジューラを特定の設備の必要に応じて調整する任意機能がある．この例は次のようなものがある：

1. 自動的順位群設定（APG）——システム利用率を平均化させるために，入出力の多いジョブの処理を優先して，優先順位をジョブに対して動的に割当てる．
2. 割当て時間——ジョブ群はラウンド・ロビン方式で一定の割当て時間ずつ処理装置を共用する．このようにすると，高順位ジョブの1つが処理装置を占用して，他の高順位ジョブが長時間実行されないようなことを防げる．

9.2.4 装置管理

たいていの360/370オペレーティング・システムには，ジョブ処理の効率を

改善することができるように，各種の装置管理機能を指定できる選択機能がある．これには次のものがある：

1. **チャネルの分離と同類（SEP/AFF）** 利用者は特定のデータ・セット（ファイル）が同一あるいは別個のチャネルによって制御される装置に割付けられるよう指定することができる．たとえば，利用者が，2つのデータ・セットは同時にアクセスされるということを知っていれば，これらに別のチャネルを割当てるよう指定する．
2. **入出力装置またはボリューム** 利用者はデータ・セットを格納すべき装置の種類または特定の装置を指定することができる（たとえば，磁気テープ17というように）．
3. **割付け** データ・セットは動的に容量を増すことができる．利用者は，最大16個まで，連続しない部分に領域をとることができる．利用者は第1領域の容量を主容量として指定する．残りの15領域は必要に応じて結合される．その容量は副容量として指定される．
4. **同期的誤り処理** 利用者は入出力誤り処理プログラムを指定することができる．
5. **バッファ群** 利用者はバッファについて次のような指定を行なうことができる：

   BUILD： 静的——バッファは利用者プログラムの中に作られる．
   GETPOOL： 要求による——バッファはシステム領域に作られる．
   AUTOMATIC： 必要に応じてシステムがバッファを割付ける．

6. **バッファの割付け** 利用者はダブル・バッファリングや動的バッファリングの指定を行なうことができる．
7. **連鎖したスケジューリング** 利用者は回転位置検出の指定を行なうことができる．（システムは，シークの順番の再編成を行なう．）
8. **スプリット・シーク** 利用者は，入出力を行なう前に予備シークをシステムに要求する．（370のブロック・マルチプレクサ・チャネルでは，シークがチャネルを専有しないので，この機能は不要である．）

これらの装置管理の機能に加えて，利用者は次に示す，情報管理に深い関係のある機能を指定することができる：

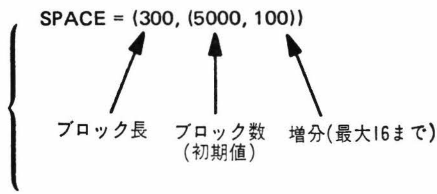
1. レコードの書式（RECFM）——固定（F），可変（V），不定（U）
2. レコード長（LRECL）

3. ブロック長（BLKSIZE）
4. データ・セットの記号名（DDNAME）
5. データ・セットの編成（DSORG）——順編成（PS），指標つき順編成（IS），分割編成（PO），直接アクセス（DA）

9.2.5 情報管理

9.2.5.1 領域割付け

利用者がどのようにしてファイルの割付けを行なうかを考えてみよう．利用者は，装置の種類に関係なく次のような指定を行なうことができる：



または，装置の種類によっては次のようにシリンダ指定によって行なうこともできる：

SPACE＝(CYL, (3, 1))，UNIT＝3330

9.2.5.2 ボリューム内容表（VTOC）

システムはどのようにしてファイルを管理するのであろうか．各ボリュームにはボリューム内容表 VTOC が設けられている．この中の各欄は44バイトのキーと96バイトのデータを持っている．欄には次の6つの型式がある：

1. データ・セットと最初の3つの増分に関する情報（これ以上増分のあるときには型式3の欄へのポインタ）
2. 指標つき順編成のデータ・セットに対する付加情報（型式1の欄から指される）
3. 最大13増分の情報
4. VTOC 欄（増分なし）
5. 空き領域（26個の空き領域に関する情報）

6. 共用シリンダの割付け情報

型式6の欄を用いることによって，同一のシリンダを2つのデータ・セットの間で分割して使用することができる．たとえば，入力ファイルの一部分と出力ファイルの一部分を同じシリンダに置いてアクセスすることが可能で，能率のよい入出力を行なうことができる．

9.2.5.3 カタログ

1つのファイルが創られたとき，どのボリュームにファイルが格納されているかをシステムはどのように管理するのであろうか．特別のボリューム SYSRES には，**カタログ** という特別のファイルが納められている．ファイルの名前とボリューム番号はこのカタログに格納することができる．

9.2.5.4 分割編成データ・セット

OS では，分割編成データ・セットとして，小さなファイルから成るライブラリを作ることができる．図9.8はOSライブラリの例を示している．このライブラリは1つのデータ・セットから成っており，このファイルの最初の部分

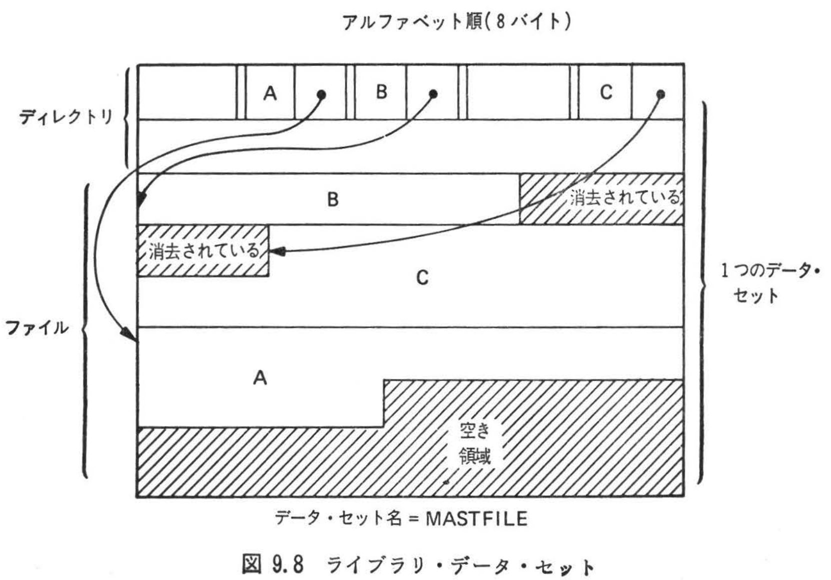


図 9.8 ライブラリ・データ・セット

はディレクトリとなっている．データ・セットの1つの要素（この場合ファイル）は MASTFILE(A) のように参照することができる．

OS にはライブラリの要素を削除する機能があるが，その結果生じた'穴'の管理は行なわず，この穴の再使用はできない．OS では，ライブラリで領域を使い果たしてしまうことがよく経験される．たとえば，Bが削除されて新しく1つのBが追加される度に，データ・セットの容量は増加する．利用者が穴を除去しようとするときには，全ファイルを複写して旧ファイルを削除する必要がある．DOS および IBM 1130 ディスク・モニタでは，一定の条件（たとえば穴が50%となる）となると分割編成のデータ・セットの中のファイルをすべて再配置して，自動的に穴を削除する機能が備わっている．

9.2.5.5 アクセス法

OS では，ファイルに対して2とおりのアクセス法を提供している：

1. レコード・レベルのアクセス QA： この型のアクセスでは GET/PUT マクロを使用する．これらは順編成アクセスに対して使用される．レコード・レベルのアクセスでは，システムが自動的にブロッキングやデブロッキング，バッファリング，および入出力の同時実行を行なう．
2. 基本アクセス BA： この型のアクセスでは READ/WRITE マクロを使用する．これらは順編成および直接編成のアクセスに対して使用される．直接編成のアクセスの場合には，利用者は次の値のいずれかを指定する．

   a．相対レコード番号（固定長レコードの場合のみ）
   b．相対トラック番号とキー
   c．相対トラック番号とレコード番号
   d．実番地

基本アクセスの場合には，システムはレコードではなくて物理ブロックの処理を行なう．ブロッキング，デブロッキング，および同期は利用者が行なわねばならない．すなわち，利用者は入出力完了の検査を行なう必要がある．(CHECK マクロを使用することができる．)

9.2.5.6 ディレクトリの階層

要約すると，OS 360/370 のファイル・システムは限定された階層構造で動

作している．この階層の要素は SYSRES ボリュームの VTOC, カタログ，ライブラリ，ライブラリの要素，およびデータ・セットである．図 **9.9 a** はファイル・システムの論理構造を，図 **9.9 b** はこれらのファイルの物理的な格納方法を示している．

9.3 コンパチブル・タイムシェアリング・システム (CTSS)

9.3.1 体系と歴史

コンパチブル・タイムシェアリング・システムは1961年に原形が発表され(Corbató, 1962),（1963年に M.I.T. で一般に使用されるようになり(Crisman, 1965), 1973年に終了した．CTSS は最初のタイムシェアリング・システムの１つであり，単純で歴史的に重要であるのでここで説明する．（それ以後のシステムの多くは CTSS の設計に基づいている.）このシステムには，重要な資源管理技法が採用されており，それらの内のあるものは未だ他では真似されていない．また，このシステムは他の計算機で容易に組込むことができる．

CTSS は一部改造した IBM 7094 で構成された．通常はタイムシェア・モードで30人の利用者が使用でき，FORTRAN モニタ・システム (FMS) によって１つのバッチ・ジョブ流を実行できる．普通の FMS バッチ処理システムで実行するプログラムは CTSS でも実行できるという意味で，両立可能（コンパチブル）なシステムである．タイムシェア・モードにおいて，各利用者はファ

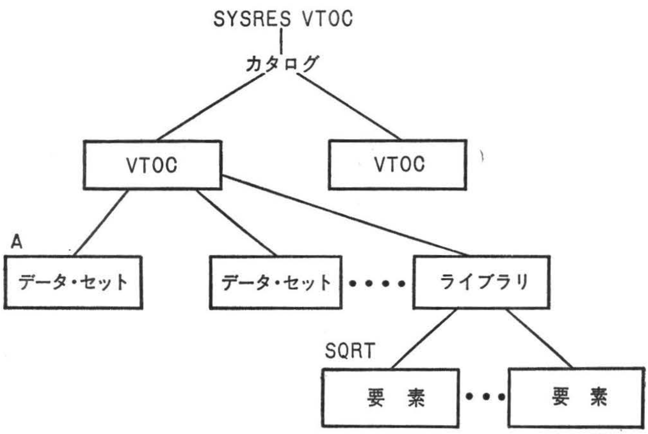


図 **9.9 a**　OS 360/370 の限定された階層

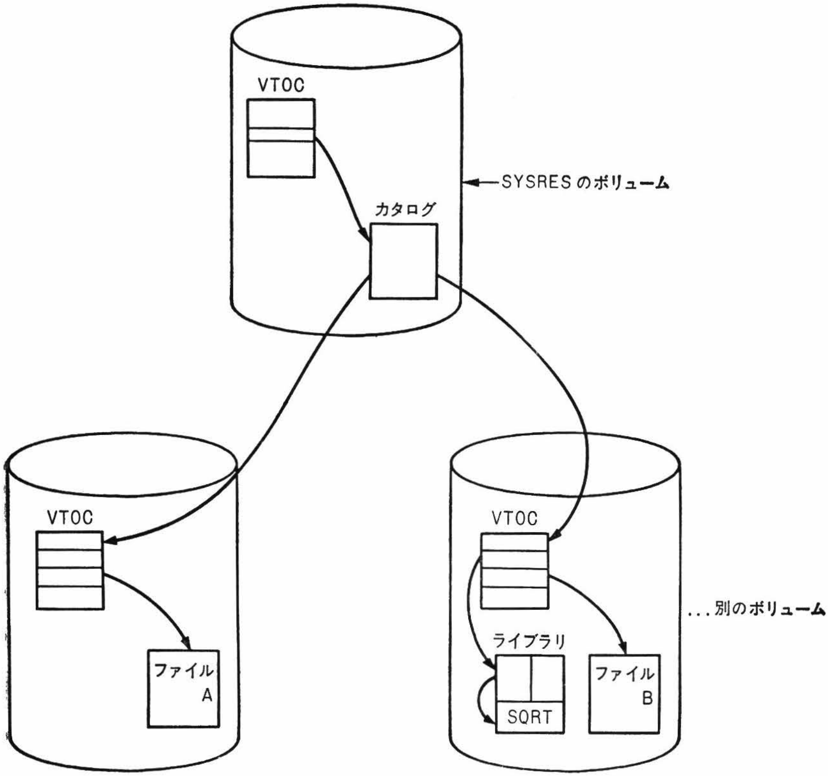


図 9.9b　360/370 のファイルのための物理的な記憶装置

イルを作って格納することができる．利用者はこれらのファイルやシステム・ファイル（コンパイラ，アセンブラ，エディタなど）の実行を行なうことができる．ファイルの入力に対しては，利用者はライン・エディタ，コンテクスト・エディタ，および汎用のマクロ・エディタなど広範なエディタを選択することができる．

CTSS は，MULTICS システムの多くのソフトウェアを開発するのに使用され，CP/CMS システムの開発の初期に重要な役割りをになった．このシステムはソフトウェアの開発経費と時間の減少，文書化の質の向上，およびプログラム効率向上の面で，ソフトウェア工場の手法（ソフトウェア開発の道具）が

```
 login m1416 madnick
W 1315.1
PASSWORD
 PARTY LINE BUSY, STANDBY LINE HAS BEEN ASSIGNED
 M1416 2286 LOGGED IN   1/10/66   1315.6 from 20000K
 CTSS BEING USED IS     DMN2C5

LAST LOGOUT WAS 1/6/66 1756.0
R 6.783+750


edl simple mad
W 1316.4
 FILE SIMPLE  MAD NOT FOUND.
INPUT
normal mode is integer
floating point a
       print comment$numbers,pleeuhz''''''''ase$
       read data
       a=sqrt(b+c)
       d=bc
end of?          end of program

EDIT
t
l mode
NORMAL MODE IS INTEGER
r      normal mode is integer
n
r      floating point a,d
l a
       PRINT COMMENT$NUMBERS,PLEASE$
l a=
       A=SQRT(B+C)

c /t/t./
       A=SQRT.(B+C)
l d=
       D=BC
i
INPUT
       print results a,d
       execute exit.


EDIT
file
R 5.833+4.250


print simple mad
W 1321.3

SIMPLE    MAD    01/10  1321.4

       NORMAL MODE IS INTEGER
       FLOATING POINT A,D
       PRINT COMMENT$NUMBERS,PLEASE$
       READ DATA
       A=SQRT.(B+C)
```

（login m1416 madnick ← 利用者が打鍵）

（PARTY LINE BUSY, STANDBY LINE HAS BEEN ASSIGNED ← CTSSが印刷）

（edl simple mad から R 5.833+4.250 まで：SIMPLEという名前のMADプログラムを作成して修正している）

図 9.10 CTSS の会話例 (第1部)

```
       D=BC
       PRINT RESULTS A,D
       EXECUTE EXIT.
       END OF PROGRAM
R .616+416

mad simple
W1321.9
 THE FOLLOWING NAMES HAVE OCCURRED ONLY ONCE IN THIS PROGRAM.
 COMPILATION WILL CONTINUE.                                        } SIMPLE完了
       BC
       B
       C
LENGTH 00072.  TV SIZE 00006.  ENTRY 00016
R 2.766+.533

edl simple mad
W 1322.8
Edit
l bc
       D=BC                  } プログラムを訂正する
c /b/b*/
       D=B*C

file
R 3.516+1.450

mad simple
W 1323.7
LENGTH 00071.  TV SIZE 00006.  ENTRY 000015  } SIMPLEを再びコンパイルする
R 2.216+.750

loadgo simple
W 1324.1
EXECUTION.
NUMBERS,PLEASE                                      } SIMPLEプログラム
b=7,c=2*                                              をロードして実行する


       A = 2.707999E 26,           D = 14.000000
  EXIT CALLED. PM MAY BE TAKEN.
R 6.166.1.050

edl simple mad
W 1325.5
EDIT
l mode
       NORMAL MODE IS INTEGER
d
l read
       READ DATA
l      whenever (b+c).l.0.,transfer to tag
l exit
       EXECUTE EXIT.
```

図 9.10 CTSS の会話例（第2部）

```
c //tag 2/
TAG2   EXECUTE EXIT.
i
INPUT
tag    print comment$negative argument$
       transfer to tag2

EDIT
file
R 2.950+3.150

mad simple
W 1523.1
LENGTH 00107.  TV SIZE 00006.  ENTRY 00020
R 2.966+.900

loadgo simple
W 1523.7
EXECUTION.
NUMBERS,PLEASE
b=7.,c=2.*

       A =  3.000000, D = 14.000000
  EXIT CALLED. PM MAY BE TAKEN.
R 6.566+1.083

loadgo simple
W 1524.7
EXECUTION.
NUMBERS,PLEASE
b=-7.,c=,2.*
NEGATIVE ARGUMENT
  EXIT CALLED.  PM MAY BE TAKEN.
R 6.216+.816

delete simple *
W 1526.1
R 1.716+366


logut
W 1528.2
 LOGUT NOT FOUND.
READY.


logout
W 1528.4
 M1416  2286 LOGGED OUT  1/10/66   1536.3 FROM 20000K
 TOTAL TIME USED=    00.7 MIN.
```

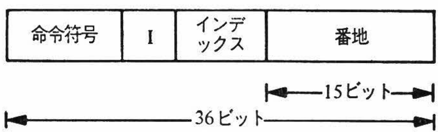
図 9.10 CTSS の会話例 (第3部)

きわめて有効であることを示した．ソフトウェア開発には CTSS のようなシステムの利用が必要であることを力説したい．

図 9.10 は CTSS の端末における会話の例を示している．利用者と CTSS とはテレタイプや電動タイプと類似の端末装置を用いて会話する．この会話で利用者入力は小文字で，CTSS出力は大文字で印刷されている．この会話例では，1つのプログラムがCTSSのファイル・システムに格納され，変更され，コンパイルされ，数回試験されている．このプログラムはMAD言語で書かれており，利用者のコンソールから2つの数を読込み，和の平方根と積を計算して出力するだけのプログラムである．

9.3.2 ハードウェア

第9.3.1節で述べたように，CTSS は改造された IBM 7094 に組込まれた 7094のハードウェアは次の特徴を有している：



主要な変更点は第2の32Kの主記憶装置（図 9.11）および再配置と限界レジスタである．すべての命令は15ビットの番地部を有し，64Kの番地を指定するにはビットが不足であるので，A，B記憶装置を指定するのに1つの番地指定ビット・レジスタが追加された．A記憶装置はオペレーティング・システムのために使用され，B記憶装置は主として利用者プログラムのために使用される．7ビットのレジスタが2つ，再配置および限界レジスタとして追加された．(限界および再配置の指定は 256 語のブロックで指定される．) 限界レジスタは

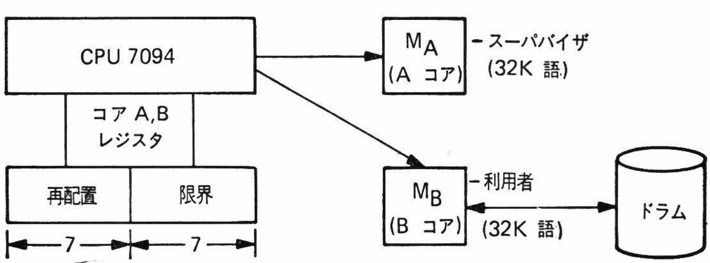


図 9.11 CTSS のハードウェア構成

利用者プログラムが番地空間の外をアクセスしないように保護するために使用することができる.

1台のドラムが利用者プログラムをスワップ入出力するのに使用される. 32KのB記憶装置全体は約250ミリ秒でスワップすることができる.

本書の CTSS の説明では, 利用者とプロセスとジョブは同一である.

9.3.3 記憶管理

CTSSで用いている基本的な記憶管理アルゴリズムは, たまねぎの皮のスワッピング・アルゴリズムと呼ばれている. CTSSは多重プログラミングを行なっていない. 1人の利用者はB記憶装置の0番地から始めて必要なだけの記憶域を与えられる. 最初の必要量はプログラム容量によって決定される. 表やデータ領域のために追加の記憶を実行中に要求することができる. どのような場合でも, 各利用者に記憶域の限界が定められている. この値は, 使用している記憶番地の最大値である. 別の利用者を実行しようとするときには, 現在実行している利用者のプログラムとデータをスワップ出力する必要がある. スワッピングのオーバヘッドを最小にするために, 次の利用者をロードする記憶域の分だけがスワップ出力される.

図 9.12 は4人の利用者による記憶域の利用状況を示している. 利用者3がスワップ入力させるとき利用者2のすべての領域をスワップ出力する必要がない. したがって, その後で利用者2がスワップ入力されるなら, その番地空間の大部分はすでに記憶に入っている. 時刻5では, 利用者4をスワップ入力するために, 利用者1と利用者2および3の残りの部分をスワップ出力しなけれ

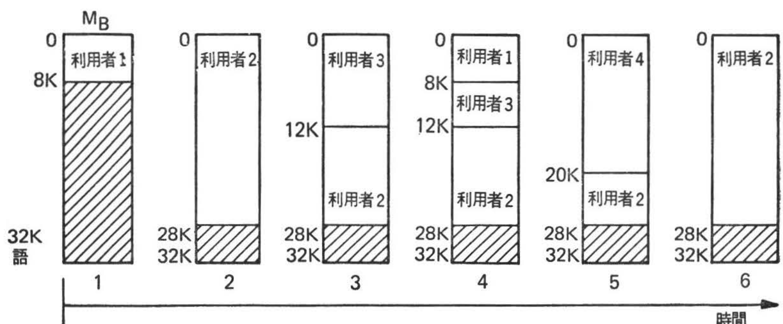


図 9.12 "たまねぎの皮" のアルゴリズム

ばならない.

すべてのジョブはB記憶装置の0番地から開始しているので，CTSSでは実際には再配置レジスタは使用されなかった．このレジスタは多重プログラミングを行なったり，スワップ入力と実行とを同時処理するために利用できるが，これらの機能は備えられなかった．限界レジスタの必要性は何であろうか．それは，記憶装置に他の利用者のジョブの部分が入っていることがあり，これを保護する必要があるからである．

9.3.4 処理装置管理

1人の利用者は次に示したいずれかの事象が起こるまで処理装置を割当てられる：

割当て時間を使い果たす．
誤りが発生する．
完了する．
コンソール入出力を実行する．

利用者が実行している入出力がすべて完了するまではスワップ出力できないことに注意してほしい（コンソール入出力を除く）．CTSSは200ミリ秒ごとに処理装置の制御を得て，コンソール入出力が完了したかあるいは他の利用者を実行すべきかを判定する．図9.13は各種のプロセス状態を示している．状態には次のものがある：

提出——利用者がデータ・セットのダイヤルを回すことに対応する．
ログ・イン——利用者はダイヤルを回したが，まだ資源は割当てられていない．
作業中——プロセスが実行可能である状態に対応する．スワップ入力され，処理装置を割付けることが必要．

CTSSでは幾種かの待合せ状態が存在する．これらの状態は待合せを行なう事象に関係する．これらは次のようなものである：

入力待合せ——バッファが空のときに，コンソールから入力バッファに入力しようとしている．
出力待合せ——コンソール出力バッファ領域を一杯にしている．出力が印刷されるまで待合せ状態となる．
ファイル待合せ——ロックされた共用ファイルの読出しを待合せてい

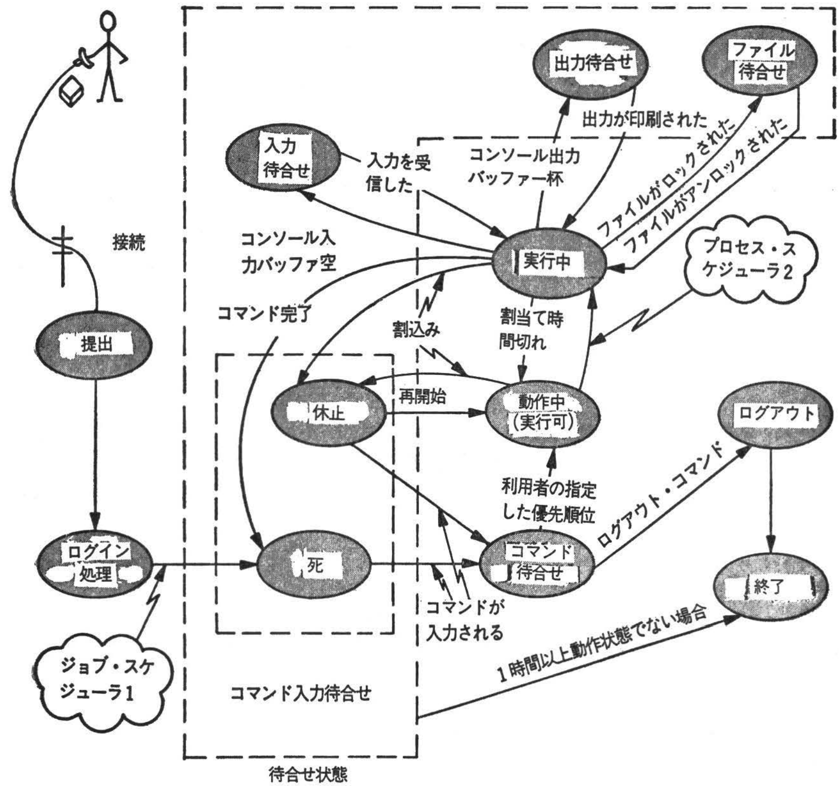


図 9.13 CTSS 中のプロセスの状態図

る．

コマンド待合せ——コマンドが入力され，動作状態となる優先順位を割当てられるのを待合せている．

休眠中——ジョブはコマンド実行の途中で割込まれている．ジョブは割込まれた点から実行を再開したり，新しいコマンドを受付けることができる．

静止——前のコマンドを完了し，次のコマンドを待合せている．

ディスクや磁気テープ入出力閉塞の状態は存在しない．ディスク入出力を実行するときにジョブは閉塞状態とならない．すなわち，ディスクや磁気テープ

入出力を実行している間は CPU を割付けられたままとなっている．一時には主記憶中に1つのジョブだけが入っているので，通常の多重プログラミングによって入出力を同時に実行することは不可能である．

9.3.4.1 ジョブ・スケジューラ

図9.13の雲形1はジョブ・スケジューラを示している．CTSS では，このスケジューリングはきわめて単純である．同時にシステムに接続できる利用者の最大数が定まっている（たとえば，30)．したがって，最初にログ・インした30人の利用者が受付けられる．システムにすでに30人の利用者が接続されているときには，他の利用者は受付けられない．他の利用者がダイヤルを回しても，システムは電話を切ってしまう．ログ・インしようとして失敗した利用者に関しては何の情報も残されない．

この単純なジョブ・スケジューラは，システムの負荷を効果的に制限するが，各種の反社会的な行動をも引き起こす．たとえば，10人の学生が1つのプロジェクトで仕事をしているとする．10人の学生は，それぞれ別の日に，月に2回の割当てで早朝（たとえば，7AM）に仕事を始める約束をする．1人の“早起き鳥”が来て，10台の端末をログ・インし，これらの端末を自分のグループのために1日中ログ・インしたままにしておく．

ジョブ・スケジューラは，これらの問題に対処できるように拡張された．すべての利用者はグループに所属させられる（たいていの利用者は M.I.T. の研究グループに属している)．各グループは1次回線数を割当てられている．上述のグループの学生には2本の1次回線が割当てられているとしよう．このグループの最初の2人の利用者がログ・インするときこの1次回線が割当てられる．システムが全負荷となったときには，このグループからはこれ以上ログ・インすることができない．一方，システムが全負荷でない場合には，このグループから3人以上の利用者がシステムにログ・インすることができる．これらの利用者には**2次回線**（**予備回線**とも呼ばれる）が割当てられる．

1次回線を使用できるグループの利用者がログ・インしようとすると，システムは予備回線の利用者を選んで，自動的にログ・アウトする．

ログ・アウトする利用者を選ぶアルゴリズムは，ログ・アウトを欲していない利用者を破壊しないようにするために，ログ・アウトする利用者を定期的に変更する．1つのシステムでは，この時点で最大の CPU 時間を使用した利用者をログ・アウトした．この正当性は機会均等という点である．（利巧な利用者は使用時間に気をつけていて他の利用者よりも多くの時間を使用すると，も

う一度ログ・インして，使用時間を0に設定し直した.）

負荷を最大30人の利用者に制限するほかに，ジョブ・スケジューラは，ログ・イン時間を2分とした．この時間内に，利用者が正しくログ・インしないと，この利用者は切断されてしまう．

このスケジューリング・アルゴリズムで最もスリルのあるのはこの効果が予想できない点にある．CTSS に基づいて1つの例を説明しよう．30人という制限はどのようにして決定されたのであろうか．各利用者はある程度の負荷をシステムに与えると仮定している．利用者の動作は一定でないから，この制限は時として控え目過ぎたり過度であったりすることがある．システム全体の負荷を監視する特別のジョブ・スケジューラが開発されたと報告されている．このジョブ・スケジューラは，負荷が一定の限度を越えると，1人の利用者を自動的に終了させて負荷を減少させる．

現在の負荷が限度を越えたとして，この負荷平衡の方式がどのように働くかをみてみよう．

1人の利用者を終了させるためにログ・アウト手続きが開始される（ファイルを閉じ，会計情報を更新する)．ログ•アウト手続きは完了するのに数秒かかる．このログ・アウトが進行中に，負荷平衡のアルゴリズムは負荷を再び検査して，負荷が実際に増加していることを見出す．（これはログ・アウト手続きが集中的な負荷を与えることから起こる.）この結果別の利用者が選択されて終了させられる．これによって負荷が増加し，別の利用者が終了される．負荷平衡アルゴリズムが満足されたときには，すべての利用者が終了される．このアルゴリズムが実際に使用されたかどうかはわからないが，“利巧”なスケジューリング・アルゴリズムで起こり得る危険な状態を示している．

9.3.4.2 プロセス・スケジューラ

図9.13の雲形2はプロセス・スケジューラを示している．幾つかのアルゴリズムが試されたが，多くはきわめて複雑なものであった(Scherr, 1965)．CTSS のアルゴリズムの目標は，大規模な計算を行なう非会話型のジョブを犠牲にして，会話型の利用者に迅速に応答することである．

このアルゴリズムは，8つの実行可行列(0〜7)を作ってその目標を果たす．会話型利用者は小さい番号の行列に，計算限界のジョブは大きい番号の行列に入れられる．プロセス・スケジューラは番号の小さい行列の中のジョブを先に実行する(会話型ジョブを優先させる)．番号の大きい行列の中のジョブには大きな割当て時間が与えられ，大規模な計算限界のジョブのスワップ時間のオー

バヘッドを減少させている．各行列のジョブには割当て時間が与えられるが，その量は行列番号の関数である．基本の割当て時間 Q は500ミリ秒である．n をレベル番号として

$$割当て時間 = Q \times 2^n$$

最初すべての利用者はレベル2または3から開始する．

ジョブが4K語より小のとき，レベル2から開始する（割当て時間＝2秒）．

ジョブが4K語以上のとき，レベル3から開始する（割当て時間＝4秒）．

ジョブが現在の割当て時間で終了しないときには，次のレベルに移る（4,5,6,7）．

優先権先取りは，現在実行しているジョブが優先権を先取りしようとするジョブの割当て時間よりも長い時間実行しているときに行なわれる．たとえば，レベル7が実行していてレベル3のジョブが実行可となっても，レベル7のジョブがレベル3の割当て時間よりも長く実行されるまではレベル3のジョブはスワップ入力されない．この方針は，ジョブがスワップ入力された直後に優先権を先取りされることがないようにするものである．ジョブのスワップ時間は，スワッピング・ドラムの速度が比較的遅いので，CTSSではきわめて重要なことである．

ジョブ行列の処理で支配する法則にはこのほかに幾つか存在する．利用者ジョブがテレタイプからの入力を受取ったときには，このジョブは会話型であるとされて，レベル2となる．さらに，実行可状態のジョブが1分間実行していないときには，このジョブは次に番号の小さい行列に移される．たとえば，行列番号7に落ちたジョブを考えてみよう．0〜6の行列にジョブが存在する間は，このジョブは実行されることはない．このジョブが1分間実行されないときにはレベル6となる．このようにして，すべてのジョブは少なくとも8分間に，1回，500ミリ秒（番号が0の行列に達したとき）だけでも実行される．

使用されていない端末（利用者が昼食を食べに行った）を調べるオーバヘッドを減少させるために，1時間以上非実行状態のジョブは自動的に終了させられ，システムからログ・アウトされる．

システムにはオンライン利用者が開設するジョブのほかに（これらは**前景ジョブ**と呼ばれる），他の種類のジョブが存在する．1つのジョブは，バッチ処理のオペレーティング・システムとして動作するFORTRANモニタ・システ

ムを実行させる（これは背景ジョブと呼ばれる）．この背景ジョブはテレタイプからの入力を受取ることがないので，通常，前景ジョブよりも低い優先順序のレベルを割当てられる．このジョブは非会話型ジョブとして取扱われる．

このほかに幾種かの悪魔とか妖精とか呼ばれるシステム・ジョブが存在する．これらの悪魔はつねにファイル・システムを検査する．このようにして，システム・ダウンのときのバックアップを図っている．

9.3.5 装置管理

多重プログラミングが行なわれていないので，装置管理はむしろ簡単である．3種類の入力だけが実行される：

1. コンソール：　各ジョブにはコンソール用として，入力および出力バッファ領域が割当てられる．コンソール入出力はすべてA記憶装置の中のスーパバイザによって処理される．コンソールからの入力は，ジョブがB記憶装置に入っているかいないかに拘らず，A記憶装置の中のジョブ用のバッファ領域に格納される．入力が受取られると，利用者の状態が適当に変更される．実行中のジョブのコンソール出力は，A記憶装置の出力バッファ領域に格納されスーパバイザは即座にこの入出力を実行する．バッファ領域が一杯になったときには，ジョブは出力待合せ状態となる．
2. ディスク入出力：　ディスクまたはドラム入出力は，利用者のファイル読出しと記憶管理のスワップ入出力の2つの場合に行なわれる．利用者によるファイルの読出しは要求に応じて実行され，この入出力中は利用者は実行中であるとみなされる．スワッピングは記憶管理により実行される．
3. 磁気テープ入出力：　利用者は，ディスク入出力と同様にして（利用者はこの間実行中であるとみなされる），磁気テープ入出力を開始させる．印刷機出力，カード読取り機入力等は背景ジョブの FMS 利用者だけが実行する．これらの入出力はすべて磁気テープ上で行なわれてから，オフラインの計算機（1401）に移されて実際の処理を受ける．

9.3.6 情報管理

CTSS のファイル・システムはかってそうであったし，現在みても進歩したシステムの1つである．この機能は今日ですら他のシステムにはない．たとえ

ば，たいていの異状ログ・アウト条件（たとえば，システム停止時）の下で，システムは各ジョブが再実行できるように各ジョブの状態を保存する．情報管理は次の特徴を有している：

1. 利用者はプログラムとデータをディスクに作って保存することができる．
2. システム・プログラムは2次記憶に格納され，利用者のプログラムやデータと同様にして管理される．
3. すべてのファイルは記号名で参照することができる．
4. 複数のファイルを同時にアクセスすることができる．
5. ファイルへのアクセスを制限することができる（たとえば，読出し専用）．
6. 論理レコードの書式とは無関係に，標準の物理レコード書式がシステムで統一的に使用されている（1レコード／トラック）．
7. バックアップ・システムは自動的にファイルの写しを磁気テープに作成する．
8. 利用者でファイルを共用することができる．
9. 組込みと保守を単純にするために，モジュールに分けた設計が行なわれている．

ファイルの構造は図 9.14 に示したように，2階層で編成されている．主ファイル・ディレクトリは，すべての利用者のディレクトリ，システム・ディレクトリ，およびライブラリ・ファイルを登録している．各利用者ディレクトリは，各ファイルに対して1つの欄を持っている．1つのファイルの物理レコードは，図 9.14 に示したように，すべて連鎖されている．このようにすると順編成ファイル（最も多く用いられる）の処理が能率よくなり，乱アクセスは非常に能率が悪くなる．

許可と結合の情報を納めた特別のファイルは，システム内のすべてのファイルのアクセス権と，他の利用者のファイルに結合を行なっている利用者を登録している．

Madnick という利用者は

PERMIT DONOVAN ALPHA READ ONLY

というような PERMIT コマンドを出してファイルのアクセスを Donovan に許可することができる．このコマンドは，ALPHA という名前のファイルを

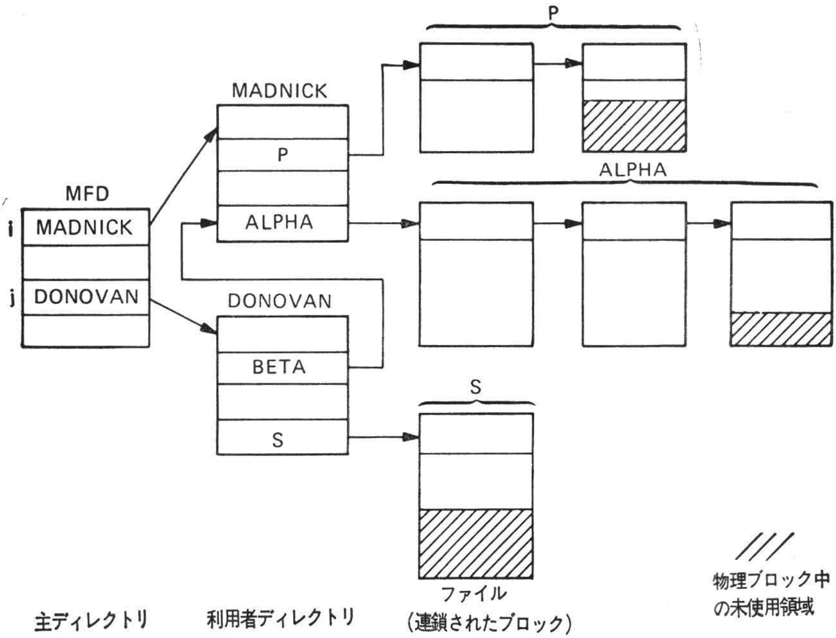


図 9.14 CTSS の階層的ファイル・システム

Donovan が読出してもよいという意味である．この許可をもらってファイルを使用するには

LINK MADNICK ALPHA BETA

というようなLINK コマンドを出す必要がある．このコマンドは，MADNICK のファイル ALPHA を自分のディレクトリの中の BETA という名前で使用したいという意味である．

各ファイルのディレクトリ項目にはファイルの“最終変更時刻”と“最終複写時刻”の情報を納めている．前にバックアップの悪魔と呼んだシステム・ジョブは，たえず，すべてのディレクトリの中で，最終変更時刻が最終複写時刻よりも大きいファイルを探している．このようなファイルを見つけると，これを磁気テープに複写して最終複写時刻を更新する．ファイルが利用者の誤りやシステムの故障で破壊されたときでも，写しをこのバックアップ・テープから

求めることができる.

9.4 多重化情報計算システム (MULTICS)

9.4.1 体系と歴史

多重化情報計算システム (MULTICS) は M.I.T. の MAC 計画[1], Bell 電話研究所, および General Electric 社[2] が協同で開発したものである. MULTICS の開発は MAC 計画で1964年に開始され, 1969年以来 M.I.T. で一般に利用されている. MULTICS の計画は, 6編の報告として1965年の秋期連合計算機学会 FJCC に発表されている (Corbató など, 1965). 主たる目標は**コンピュータ・ユーティリティ**の実際の原型を開発することであった. コンピュータ・ユーティリティとはタイムシェアリングの概念の一般化であり, 電気や電話と同様に遠隔の利用者へ中央の計算機の資源を効率よく分配することができるものである. 情報の安全性と個人の秘密のために特別の注意が払われている. 本書では MULTICS から幾つかの例を示しているので, 本節では興味のある部分だけを詳細に説明することとする.

システムを設計して開発することのほかに, この計画の目標の1つは, 優れた設計と組込みの基礎となっている概念を明確にすることであった. システムの開発に当っては, MULTICSの組込みで採用された3つの設計技法に注目すべきである (Corbató, 1969):

1. 組込む前に設計すること.
2. 組込みにタイムシェアリング・システムを使用すること (MULTICS の場合には CTSS が使用された. 現在 MULTICS は Honeywell 情報システム社で次システムの組込みのために使用されている).
3. 機械に独立な機能の組込みには高水準言語 (PL/I など) を使用すること.

MULTICSはその発端からしてきわめて意欲的なものであった. 設計に多数の新しい概念や, 以前には限定された方法でしか使用されていなかった概念を

1) MAC 計画 (MAC は人間と計算機, 多重アクセス計算機, あるいは計算機援用の認識を意味している) は M.I.T. の大きな研究グループの1つであり, 多方面の計算機利用を研究している. MAC 計画は M.I.T. の CTSS の開発にも参画した.
2) このグループは現在 Honeywell 情報システム社に吸収されている. GE 645 として知られた MULTICS の最初のハードウェアは現在は HIS 645と呼ばれている.

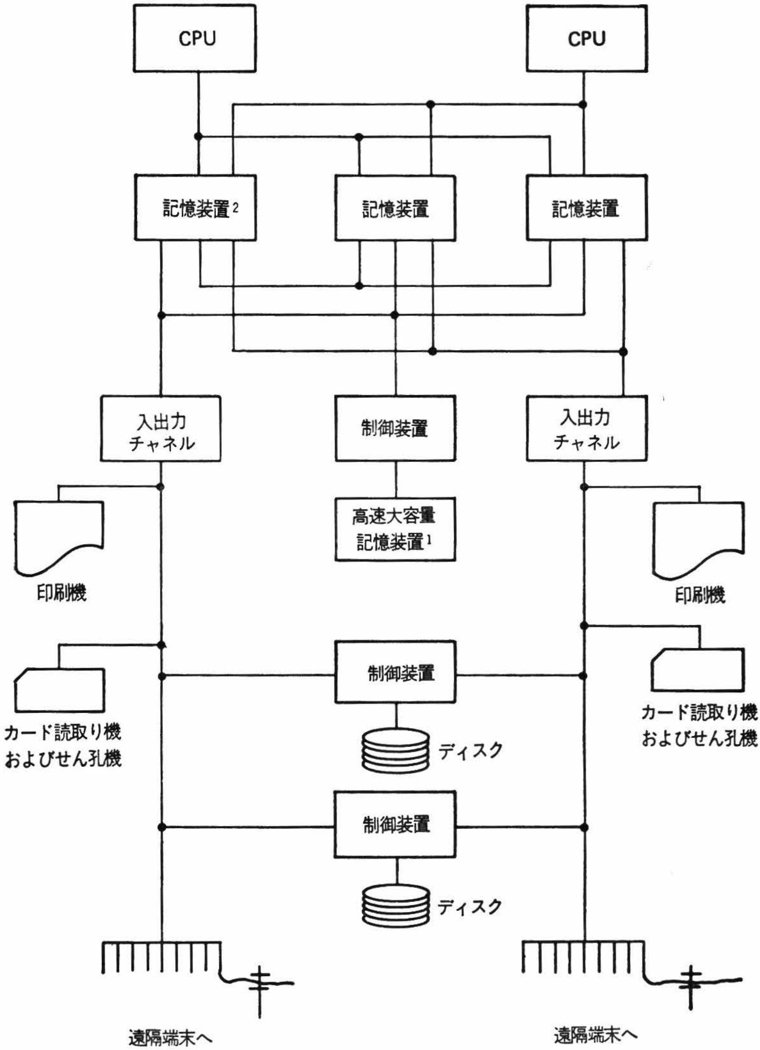
多数採用している．これらはきわめて多いので，顕著な特徴をすべて識別することは困難である．最も著しい成果の1つはページングとセグメンテーションによって，記憶管理と情報管理の多くの機能を一体化したことである(第3.7.3.8節および6.11節参照)．参考文献，特に Organick (1972) と Corbató (1972) ではこれらの詳細を豊富に示している．

9.4.2 ハードウェア

最初の MULTICS は GE 645 (現在 HIS 645) を使用した．この計算機は GE 635 システムを特別に改造したものである．進歩した計算機ハードウェア製造技術と MULTICS を使用して得た経験を生かして，HIS 645 は 1973年に新しい HIS 6180 に置換られた．本書で説明するハードウェアの特徴の大部分は HIS 645 と HIS 6180 の両方に適用する．

HIS 6180 のハードウェアは，図 9.15 に示したように，1台以上の CPU で構成されている．記憶装置は36ビット語で，第3.8章に述べたように，ページングとセグメンテーションのハードウェアを備えて仮想記憶を構成している (Bensoussan, 1969)．他のシステムで同様のデマンド・ページング (Fotheringham, 1961) とセグメンテーション (Burroughs, 1961) が開拓された．この手法は別の新しいシステムのハードウェアでも使用された(Comfort, 1965；Motobayashi, 1969)．ハードウェアの詳細は Honeywell のマニュアルやもっと詳しくは Organick (1972) に示されている．本節では特徴的な事項のみを述べる．

記憶装置は (名前，オフセット) の順序対で番地指定される．ここで“名前”はセグメントを示し，“オフセット”はこのセグメント内の語番号を示す．ハードウェアとソフトウェアは，この2次元的な番地を特定の主記憶番地に変換する．ハードウェアはプロセスごとに 2^{18} 個のセグメントを可能とする．各セグメントは最大 2^{18} 語の長さまで可能である．しかし，MULTICSではセグメント長を 2^{16} 語に制限している．セグメントは64語のページまたは 1,024 語のページに分割される．現在の MULTICS のソフトウェアは大部分のセグメントについて 1,024 語のページを使用している．2種類のページ容量を使用すると，物理的な記憶装置管理が複雑となるので廃止された．すべての番地指定は，**ディスクリプタ・セグメント**と呼ばれるセグメント・マップ表 (**ディスクリプタ・ベース・レジスタ**によって指されている) とページ表を用いてハードウェアで実行される．この型の番地指定は第3.8節に述べたように実行される．命令には2つの型式がある．型式0は同一セグメント内のデータや命令を参照する



1 高速大容量記憶装置はHIS 645ではドラムであり，HIS 6180では大容量コア・メモリである．

2 それぞれ128 K語の記憶装置が3台．

図 9.15 MULTICS のハードウェア構成の例

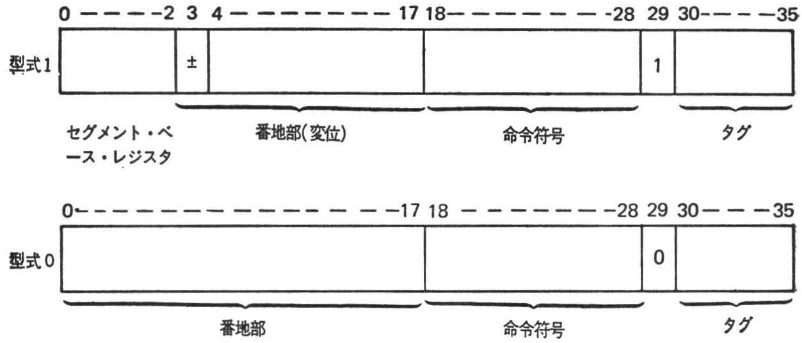


図9.16　典型的な H6180 の命令書式

ために，型式1は任意のセグメント内のデータや命令を参照するために使用される（図9.16）.

ベース・レジスタは8個あり，セグメント・ベース・レジスタ部によって指定することができる．第3.7.2.4節で述べたように，多重の間接番地指定をセグメント内およびセグメント間参照で用いることができる．Organick（1972）は番地形成の別のモードと型をも示している．

9.4.5　記 憶 管 理

記憶管理モジュールはページングされたセグメンテーションを行なう記憶参照はすべて，第3章および第6章で論じたように，ページ・マップ表をセグメント・マップ表を用いて写像させる（Corbato, 1969；Daley, 1968)，セグメントの最大長は64K語で，各プロセスは最大256K個のセグメントを持つことができる．セグメンテーションを行なうことによってダイナミック・リンキングの実現とセグメントの共用が可能となる．

9.4.3.1　MULTICS の記憶階層

今日のたいていのタイムシェアリング・システムでは，デマンド・ページングの記憶管理と情報管理とは別個に取扱われている．通常，“ページング・ドラム”などの高性能の装置が主記憶に置かれない利用者の番地空間のページを収容し，可動腕型ディスクなどの低速であるが容量の大きい装置が利用者のファイルを収用するために使用されている．MULTICSでは，すべてのファイルをセグメントとして取扱い，すべての“ファイル入出力”をデマンド・ページン

グの機構によって実行している．これらの間にははっきりした区別が存在しない．

このようにセグメントとファイルを同一視する考えは大変概念的には優れているが，これを実際に組込むには幾つかの問題点がある．もし可動腕型ディスクだけを使用したら，優れた性能を実現するためのページング速度が得られないであろう．（負荷の大きいタイムシェアリング・システムでは毎秒50〜100ページのスワップが必要である）．一方，高性能ドラムを専用使用することはきわめて高価につく（大規模なタイムシェアリング・システムでは4億バイト以上ものオンライン記憶域が必要である．このためには高価なドラムが50台以上必要である.）この問題を解決するために，MULTICSでは図9.17に示したように，高価で高速の装置と安価で伝達の装置とを用いる記憶の階層を構成している(Madnick, 1973)，簡単のために，主記憶，ドラム，およびディスクをそれぞれ L1，L2，および L3 記憶と呼ぶことにする．

L2 と L3 が共に存在するときには，ファイルをどこに置くべきかを決定することが必要となる．理想的には，ほとんどの入出力が L3 よりも L2 に行なわれる方が望ましい．しかし，L2 の容量が小さいのでこれを行なうことは困難である．最初 MULTICS では，次の法則によって決定を行なっていた：

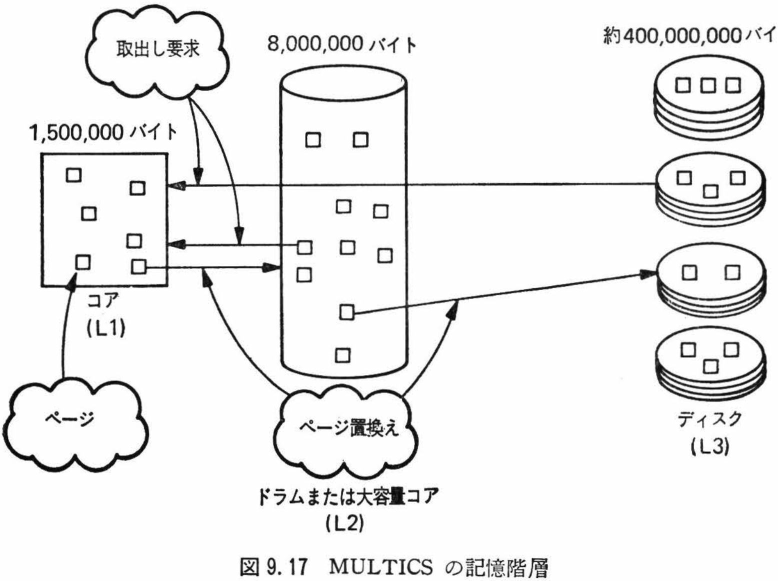


図9.17　MULTICS の記憶階層

1. すべての一時使用のセグメントとスタック・セグメントはドラムに入れる.
2. 注意深く(人手で)選択されたシステム・セグメント,特にオペレーティング・システムの非常駐の部分はドラムに入れる.
3. その他のセグメントはディスクに入れる.

このような法則によって,ファイルの約1%だけを収容しているに過ぎないけれどもページ入出力の90%をドラムで行なうことができる(Madnick, 1973).

これはかなり効果的であるが,まだ十分ではない.ドラムとディスクの平均アクセス時間をそれぞれ10ミリ秒および100ミリ秒とすると,第3.10.3.4で定義したように,実効アクセス時間は$0.90\times10+0.10\times100=19$ミリ秒となる.ドラム入出力の割合が95%になると,実効アクセス時間は14.5ミリ秒となり,約25%の減少となる.このような改良は上述の法則で欠けている次の点を考慮すると可能となる:

1. 頻繁に使用されたり長時間実行する利用者の手続きやセグメントがL1とL3の間でスワップされている.
2. L2の領域が実際には使用されていないセグメントを収容するために無駄となっている(たとえば,ログ・アウトせずに昼食に出かけた利用者の一時セグメントなど).

したがって,利用者の手続きセグメントをL2に,“不用”のセグメントをL2からL3に移しておきたい.

MULTICSでは次の法則を用いて動的に記憶階層を管理している:

1. ページ割出しが起こったとき,ページの写しがL2に入っていればL2から,入っていないときはL3からL1に移される.
2. L1が一杯でページを置換える必要があるときにはL2に移される.
   a. L2にページの写しが入っていればこれが書換えられる.(ページが変更されていない場合には,これは不要である.)
   b. L2に写しが入っていなければ,写しを作るための領域を求める.
3. 手順2bでL2が一杯ならば,L1から新しいページを移すための領域を作るために,ページをL2からL3に移す.(ページが変更されていないならば——純手続きは変更されることはない——写しはすでにL3に存在するので物理的に移動させる必要はない.)

同一の純手順きの写しが同時にL1,L2,およびL3に存在することがあ

る．

L1 から L2 へ置換えるページを選択するために変形 LRU アルゴリズムが使用される．動的な記憶域の有効性の尺度のもう1つは，L2 上でのページの滞在時間である．平均的には，L3 に戻されるまでにページは L2 に45分以上留まるという報告がある．

9.4.3.2 ページ置換えアルゴリズム

ページ置換えアルゴリズムは，第3.6.2.7節で論じたように，1ビットの“使用ビット”を用いてLRUの近似を行なっている．さらに，各プロセスごとに記憶管理は最近起こった200回分のページ割出しのリストとしてページの軌跡を記録している．プロセスが実行不適格となったとき（第9.4.4節参照），このリストを調べてどのページをポストページ（一度に置換えること）するか，また，後刻プロセスが実行適格となったときどのページをプリページ（一度にページ入力すること）するかを決定する．（プリページすべきページは記憶されて，再ロード時に主記憶にページが存在しないときにはプリページングが行なわれる．）ポストページとプリページを行なうかどうかの判定は，次の6項目によって，ページ軌跡の中の各ページについて行なわれる：

1. このページは未だ主記憶に入っているか．
2. このページは使用されたか（ページ表の欄の参照ビットがオン）．
3. このページは変更されたか．
4. このページは共用されない部分か．
5. このページはページング・ドラムから入力されたものか．
6. このページは現在の適格期間中に使用されたか．

この結果次のような処理が行なわれる：

1. このページをポストページする．
2. 必要ならば後刻このページを用意する．
3. このページがワーキング・セットに入っているものとみなしておく．
4. 記憶域が必要となったときこのページが置換えの候補となるように設定する．

これらは様々に組合せて実行される．実際の決定は，2^6 とおりの可能性に対して実行すべき処理を示した表を用いて行なわれる．

ワーキング・セットに入っていると決定されたページ数は，現在のワーキン

グ・セットの容量を示している. この数は, アクティヴ・プロセス表 (APT) のこのプロセスの項目の中に記録されている.

計算の内容が大幅に変化するときには (たとえば, 新しい段階を開始), ワーキング・セット (とその容量) も大幅に変化すると考えられる. したがって, ワーキング・セットの評価モジュールもこのような変化に速応できるようにしたい. 具体的には, hcs_$reset_working_set というシステム・ルーチンがこの目的のために用意されている. システムはこのルーチンを頻繁に使用する. コマンド・レベルに戻る度に, リスナー (コマンド・プロセッサ) はhcs_$reset_working_set を呼出す. このルーチンはページ軌跡表を調べて, 対応しているページでまだ主記憶に入っているページの使用ビットをすべて帰零する. 次に, リスナーはプロセスを閉塞状態として, 別の会話を待合せる. このようにして, プロセスが再び適格となったとき, ワーキング・セットとしてあらかじめロードされるべきすべての主要セグメント——たとえば, スタック, 複合リンケージ・セグメント, タイプライタ入出力機構とそのデータベース, プロセス間通信機能とそのデータベース (当然, リスナー自身をも含む) など——のページをきわめて都合よく参照することができる.

利用者プログラムも hcs_$reset_working_set を任意に呼出すことができる. これは計算の性質上大幅な変化がわかっている点——たとえば, コンパイラの2つのパスの間や, 再び参照されないようなページを多数参照した計算部分を実行した後——などで行なわれる. hcs_$rset_working_set を適当に用いると, 以後のページ割出しの処理で記憶域が直ちに使用できるように, 不要ページを迅速に追放して主記憶を解放することができる. また, このプロセスの主記憶利用度を低下させ, ひいてはシステム内のすべてのプロセス相互の利益となる (主記憶の利用効率が上昇して仕事量が増大する).

9.4.3.3 保護リング

第3.7節で論じたセグメンテーションの保護属性に加えて, MULTICS では**保護リング**の概念を採用している. この原理と組込みの総合的な説明は, Graham (1968) と Organick (1972) を参照してほしい.

保護リングは次の考えに基づいている:

1. "知る必要性"
2. 誤りによる被害を小さくする "防火壁"

例を用いて利用法を説明しよう. 計算機の科目の "採点プログラム" を考える.

このプログラムは，学生によって作られたサブルーチンの正しさを検査するために使用される．通常の2状態（スーパバイザ状態と利用者状態）のオペレーティング・システムでは，この採点プログラムをどこに置けばよいであろうか．もしこれを学生のサブルーチンと共に利用者状態に置くと，学生からの（事故または悪意からの）侵害をどのようにして保護することができるであろうか．採点プログラムが使用する GRADER セグメントの中の正しい番地に100点を記入するサブルーチンを作った学生がいた．これが偶然発見されたときこの学生は次のように語った．“採点プログラムをだますことの方が宿題を解くより難しかったが面白かった”と．

採点プログラムを保護されたオペレーティング・システムの中に含めることもできるが，これを実行するのが難しいうえに，採点プログラムの犯す誤りによってオペレーティング・システムが被害を受けやすくなる．

MULTICS では，この問題を2状態のシステムから階層的な n 状態のシステムに一般化して解決している．状態 i の中のセグメントは i 以上の状態のセグメントにアクセスできるが，i より小の状態のセグメントにはアクセスできない(手続きの呼出しを除く)．この保護リングの考えは，第1章で論じ，第7章の例題オペレーティング・システムで用いられた階層的なオペレーティング・システムを実現するための優れた機構を提供する．

HIS 6180 のハードウェアは8個の保護リングを提供している．リング0〜3はオペレーティング・システムで使用され，リング 4〜7 は利用者が使用できる．図9.18は，これらのリングを採点プログラムの場合にどのように利用できるかを示している．オペレーティング・システム自体も，あまり重要でないサブルーチンの誤りの影響を小さくするためにリングに分けられている．

9.4.3.4 主記憶の使用

デマンド・ページングを行なっている場合でも，オペレーティング・システムのある種の部分は，効率上の理由あるいは論理的な必要性から，恒久的に主記憶領域を割付けられている（たとえば，ページ割出しハンドラは主記憶に入っている必要がある.）図9.19は MAC 計画の HIS 645 において動作している MULTICS システムの大略の記憶域の使用法を示したものである．

常駐のスーパバイザは，スーパバイザのページングできないすべての部分から構成されている．主要なシステム表はアクティヴ・セグメント表 (AST) である．この表は，システム中でアクティヴな，すなわち主記憶にページ表の入っているすべてのセグメントに対して1つの欄を持っている．もう1つの主要

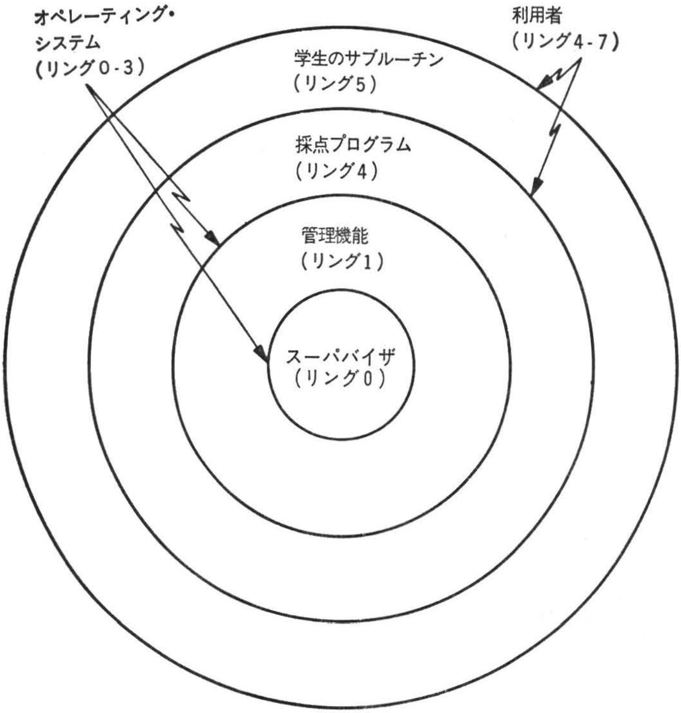


図 9.18 MULTICS における保護リングの使用

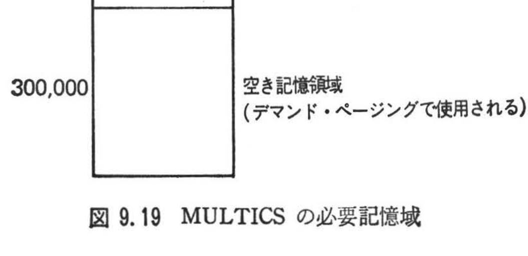


図 9.19 MULTICS の必要記憶域

な表はアクティヴ・プロセス表 (APT), すなわち, 保持および実行可プロセスのリストである.

保持状態のプロセスは APT と AST の欄に約128語必要である. プロセスが実行可, 実行あるいは閉塞状態となったとき, 主記憶の中の表の必要量は3,200語に増加する. これは主としてディスクリプタ・セグメント (セグメント表) のためのものである.

MULTICSシステムでは, 記憶管理, ファイル・システム, およびハードウェアの機能がすべて一体化している. たとえば, ファイル・システムは AST を管理しているので, すでに主記憶に入っている共用可のセグメントを必要とするプロセスの記憶域必要量を減少させることができる (純手続きは共用可能である).

9.4.4 処理装置管理

処理装置管理は, 図 9.20に示すようなプロセス状態の遷移を行なうモジュールから構成されている.

処理装置管理は2つの条件にしたがって処理を行なっている. 第1の条件は応答速度を早くしようとするものである. このために, 保持状態になることの

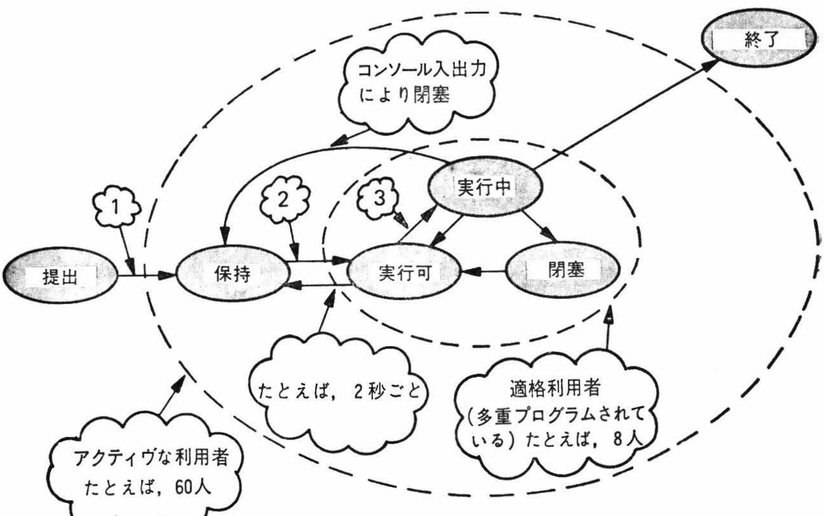


図 9.20 MULTICS のプロセス状態図

できる利用者数を制限する．任意の時点の利用者数は，その時点での負荷とシステム構成に関係する．

第2の条件はスラッシングを防止しようとするものである．これは一時に多重プログラミングされる利用者数を制限する．たとえば，60人の利用者を同時に多重プログラミングで実行するとし，記憶域が300K語とすると（図9.19），各利用者は約5,000語，すなわち，わずか5ページ（1K語のページとする）しか与えられない．この場合のページ割出しの回数はきわめて多い．一方，8人の利用者だけを多重プログラミングすると各利用者は30ページ以上を得ることができ，余分のページ割出しが起こる確率も小さくなる．

このように，処理装置は短い応答時間で“多数”の利用者間でタイムシェアを行ない，資源の有効利用の点から“少数”の利用者間で多重プログラミングを行なう．

幾つかのプロセスが一定の時間——たとえば，2秒——実行**適格**として選択され（多重プログラミングされる），その時間がたつと保持状態に戻される．コンソール入出力で閉塞された利用者は保持状態に戻される．ページ入出力で閉塞されたプロセスは適格のままである．

同時に適格となる利用者数はシステム構成と利用者の特性によって定まる．この値は通常2〜10の範囲である．

図9.20を参照して，スケジューリングの主要点を次に示す：

1. システムはどのようにして利用者を選んでシステムに入れるか（雲形1）．
2. システムはどのようにして保持状態のプロセスから適格とすべきプロセスを選択し適格とする時間を決定するか（雲形2）．
3. システムはどのようにして実行可能状態のプロセスから実行するプロセスを選択し，実行時間を決定するか（雲形3）．

これらの問に対して幾つかのアルゴリズムが試されている．問1に対する判定の基準は迅速な応答時間を保証するということである．問2に対する規準は利用率をよくするということである．問3に対して用いられた1つのアルゴリズムは，単に任意のプロセスを選んですべての資源を与え，入出力待合せを吸収するために他のプロセスを利用するものである．このようにすると，1人の利用者をきわめて短時間で完了させることができる．

9.4.4.1 負荷の制御

雲形1に対するアルゴリズムは次のようなものである．すべての利用者は次のような重みを与えられている：

通常の利用者	1単位
バッチ・ジョブ	1½単位
制限された利用者	½単位

オペレータはシステムで許される最大の単位数を設定する．この重みは使用する資源を示すためのものである．たとえば，バッチの利用者は会話処理を行なわず，コンソール入力を行なうことがないので，タイムシェアリングの利用者よりも多くの計算機の資源を使用する．M.I.T. の HIS 6180 では通常，2CPU で記憶装置が3台（各128K語）のとき，100単位，1CPU で記憶装置が2台のとき50単位である．

MULTICS利用者はすべて**負荷制御グループ**に属している．各負荷制御グループには一定の単位数が割付けられている．1つのグループの単位数が超過すると，通常そのグループの利用者はこれ以上接続することができない．もし1人の利用者が接続したとして，しばらくしてこのグループが合計の単位数を超過していれば，一定の時間の後に自動的にログアウトされてしまう．グループの責任者はこの時間を設定することができる．

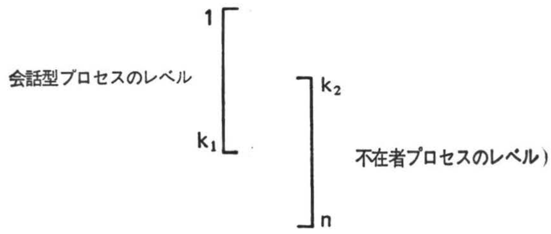
1つの制御グループの中の利用者は2つの基本的な種別――特権と非特権――に分類される．特権的利用者は非特権的利用者の優先権を先取りすることができる．この方式は CTSS で行なわれたものと類似している（第 9.3.4.1 節）．

9.4.4.2 ジョブ・スケジューラ

雲形2はジョブ・スケジーラに対応する．このスケジューラは適格とすべきプロセスを選択し，実行可状態とする．

保持リストは，各優先順位ごとに1つの行列を持つ n 個の行列の組とみなすことができる．各利用者プロセスは，最初，優先順位の範囲（$\mathscr{L}_1$, $\mathscr{L}_2$）を割当てられ，順位 $\mathscr{L}_1$ から開始する．範囲（$\mathscr{L}_1$, $\mathscr{L}_2$）は．プロセスが受けるタイムシェアリング処理の型を示している．任意の時点で1つのプロセスは $1<\mathscr{L}_1<\mathscr{L}<\mathscr{L}_2<n$ であるような優先順位 $\mathscr{L}$ を有している．

会話型プロセスは $1\sim k_1$ の範囲，不在プロセス（バッチ）は $k_2\sim n$ の範囲をもつ．ここで k_2 は，たぶん次図に示すように k_1 より高い優先順位である：



短い会話には迅速な応答を行なわせるという方針によって，処理装置を短時間使用すると予想されるプロセスに順位1が与えられる．たとえば，コンソール入力を要求して利用者が次の入力行をタイプするのを待合せている間処理装置を解放したプロセスは，入力が完了したら（直ちに）実行の再開ができることを望んでいる．このようなプロセスには順位1が割当てられる．

低い優先順位は長時間実行している会話型プロセスと**不在プロセス**が使用する．（不在プロセスはCTSSの背景ジョブと同じである．）不在プロセスに対する優先順位の範囲は (k_2, n) である $(1<k_2)$．$k_2<k_1$ の場合には，順位が $(1, k_1)$ の会話型プロセスと不在プロセスの順位には重複部分が存在する．いずれにせよ，短時間の不在プロセスより早く処理される．

最も優先順位が高い非適格プロセスのワーキング・セットが主記憶に納まる場合には（すなわち，すべての適格プロセスのワーキング・セットの予想値の和がページングを行なっている記憶容量よりも小のとき），このプロセスは適格とされる（保持状態から実行可状態となる）．適格プロセスが不適格になると空き記憶域の容量は増大する．このワーキング・セット容量の制限によって，多重プログラミングされているプロセス数はたえず変動している．

1つのプロセスが適格とされるとこのプロセスを“ロード”しなければならない．ローディングは，ディスクリプタ・セグメントを設定すること（第9.4.3.4節で述べた）と，プロセスが前回アンロードされたときに使用されていたすべてのページをプリページすることである（第9.4.3.2節に述べた）．

短いコマンドを出す会話型の利用者に迅速に応答するために，非適格のプロセスは適格プロセスの優先権を先取りすることができる．優先権を先取りされたとき，最後にスケジューリング・アルゴリズムから与えられた割当て時間で未使用の時間を持っているプロセスの優先順位 k の行列の先頭へ置かれる．

その効果は次に述べるようになる．優先権先取りを行なったプロセスが適格性を失うと，先取りされたプロセスは適格とされる．いま順位3のプロセスAが順位4のプロセスBの優先権を先取りしたとする．プロセスBは順位4の行

列の先頭に置かれる．Aが時間を使い果たすと適格性を失い，順位4の行列の最後に置かれる．Aが実行している間に別のプロセスがBの前に置かれていないならば，次にBが選ばれて実行される．

9.4.4.3 プロセス・スケジューリング

雲形3のアルゴリズムはプロセス・スケジューリングに対応する．MULTICS システムでは，実行可リストのプロセスは優先順位が割当てられている．プロセス・スケジューラは，最も優先順位の高い実行可プロセスに処理装置を与える．

各保持行列には割当て時間 q_0 が定められている．割当て時間を使い果たしたプロセスは保持状態に戻され，割当て時間 $2\times q_0$ を持つ次の順位の行列に置かれる．行列の中の位置と割当て時間に関する情報はAPTに格納されている．

実行中のプロセスがシステム事象の発生を待合せているときには，処理装置は，最も順位の高い実行可で適格なプロセスに割当てられる．事象が発生すると，これを待合せていたプロセスは実行可とされ，現在実行しているプロセスよりこのプロセスの順位が高い場合には，実行中のプロセスは処理装置を譲渡する．システムはつねに，最も優先順位の高い実行可プロセスを実行する．

プロセスがページ入出力によって待合せ状態となったときには適格性を保持したままであるから，すべての適格プロセスが同時に待合せ状態となる可能性がある．このような場合には，実行すべき適格プロセスが存在しないので，処理装置はアイドル・プロセスに与えられる．アイドル・プロセスはつねに実行可で，つねに最低の順位を持っているとみなされている．

9.4.5 装置管理

MULTICS の入出力システムは，順処理が可能なすべての入出力装置に対して，IBM のオペレーティング・システムと同様の，一般的な，装置に独立のインターフェースを用意している（これは MULTICS および PL/I の両方で**入出力流**と呼ばれている）．入出力流は遠隔端末，カード読取り機と穿孔機，磁気テープ，紙テープなどに対して使用することができる．セグメントでも順処理の入出力流の源あるいは終端とみなすことができる．

流れに対して読込み書出しを実行するには，装置管理は入出力流に対応した装置を探して，要求を適当なチャネル・コマンド語（HIS ではデータ制御語 DCW という）に変換する必要がある．この変換は，**装置インターフェース・モジュール**（DIM）と呼ばれる各装置ごとの特別のモジュールによって実行さ

れる．

入出力システムは概念的に図 9.21 のように表わされる．

HIS 645 の MULTICS システムでは，入出力は汎用入出力制御装置(GIOC)を用いて実行された．GIOC は装置スケジューラであり，ソフトウェアによって実行される．GIOC は入出力チャネルの管理と多重化を行なった．

DIM は，装置に独立の要求を装置に対応する要求に変換する．このためには，GIOC のためのプログラムをコンパイルする必要がある．コンパイルされたプログラムは，流れが接続された特定の装置の特異性を反映している．このプログラムは，遠隔端末の場合には行の制御を，磁気テープの場合には各種の制御を実行する．さらに，DIM はシステムで使用されている内部符号を装置の文字符号に変換することも行なう．たとえば，タイプライタ端末は多数の種類が存在し，それぞれは異なった文字符号を持っている．

DIM は GIOC のためのプログラムをコンパイルすると，GIOC のインターフェースの役割りを果たすモジュールを呼出す．入出力要求が完了するまで GIOC のインターフェース・モジュール (GIM) と交信するのは DIM の仕事である．このためには，GIOC がチャネルに対して用意するチャネル・プログラムの書式に応じて GIM を何回か呼出す必要がある．

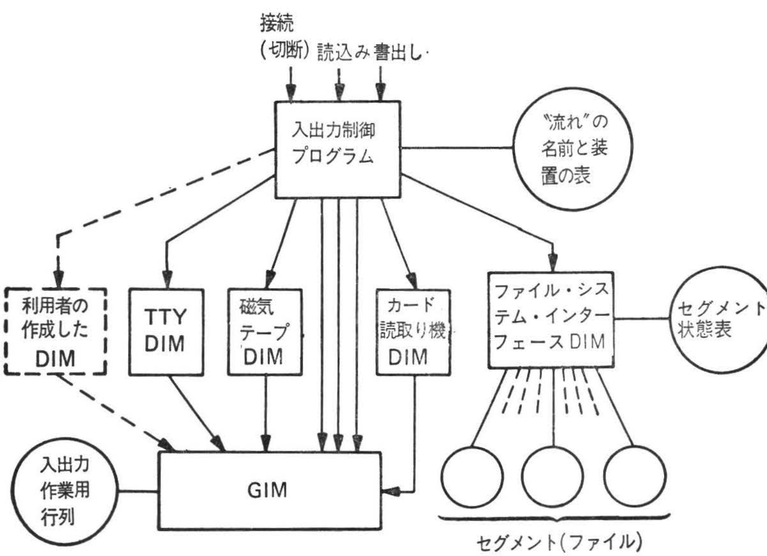


図 9.21 MULTICS の入出力システム構成の概略

GIM プログラムは GIOC 全体の管理を受持っている．GIM は割込みに応答し（GIOC の割込みハンドラである），仕事の完了を識別して DIM に適当な状態情報を送信する．

図 9.21 の入口“接続”は，接続表の中に流れの名前と装置の対応を確立させるために使用される．入口“切断”は流れの名前と装置との接続関係を無効にするために使用される．

DIM と GIOC は次の機能をも実行する：

回転位置の検出
ドラム入出力の行列管理
可能な限りの入出力同時実行
スプーリング

9.4.6 情報管理

MULTICS の情報管理では次の2つの概念が重要である：

1. ファイルをセグメントとして取扱うこと（Daley, 1968）
2. 完全な階層的ファイル構造

ファイルをセグメントとして取扱うことによって，ファイル・システムの多くの機能はセグメンテーションのハードウェアによって実行することができる（第3章および第6章参照）．階層的な構造によって，利用者に大きな融通性を与え，複雑な共用と保護，およびファイルのアクセスが可能となる（第6章参照）．

応用によっては，大規模なデータ管理システムの場合，ファイルはセグメントの最大長である 256K 語以上とすることもできる．このような大きなファイルはセグメンテーションのハードウェアだけでは完全に取扱うことができない．このような場合には，複セグメントを使用する必要がある．入出力システム（装置管理）のインターフェースを利用すると，利用者にとっては単一のセグメントでも複セグメント・ファイルでも何ら差がない．

階層的なファイル構造の全体的な構成についてはすでに論じているので，本節では MULTICS ファイル・システムの2つの機能——セグメント制御とディレクトリ制御——について述べる．

セグメント制御モジュール（SCM）は記号名からセグメント番号への写像を実行する．新しいセグメントが作られると，SCM はこれをファイルの階層の

どこに置けばよいかを決定する．セグメントが認識された後では，適当なアクセス権を与えて実行中のプロセスに既知とすることにより，SCM は保護機能をも果たす．

SCM は，プロセスの部分となっているすべてのセグメントを登録している既知セグメント表 (KST) と，アクティヴなプロセスによって現在使用されているセグメントをすべて登録したアクティヴ・セグメント表 (AST) を管理している．

ディレクトリ制御モジュール (DCM) は，ディレクトリの構成を変更するために使用される．ディレクトリ・セグメントの内容を読出したり変更したりできるのは DCM に限られているので，セグメントの状態や格納場所に関する問合せはすべて最後に DCM を呼出す．

MULTICS のファイル・システムは，前述したリング，リンク，階層構造，およびダイナミック・リンキングの概念を組込んでいる．

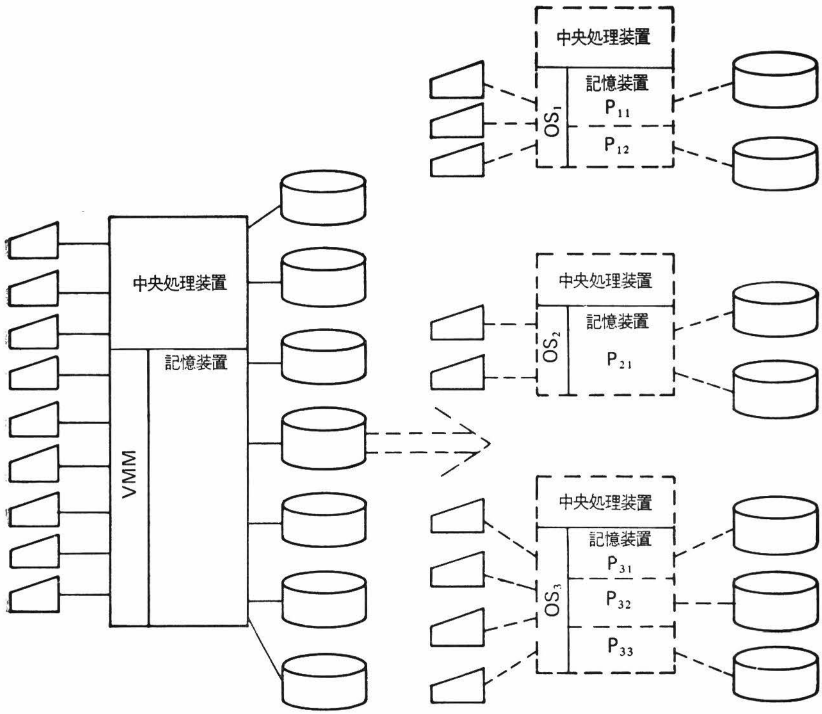
9.5 仮想計算機/370 (VM/370)

9.5.1 体系と歴史

仮想計算機/370 (VM/370) は仮想計算機モニタ (VMM) システムの1つの例である．VMM はオペレーティング・システムの1つの特殊な形であり，物理的な資源を利用者間で多重化させるだけで，その他の機能的な増強を行なわない．特に VMM によって提供された拡張計算機（第 1.5.1 節参照）は，VMM が実行している裸の計算機と同一である．このようにして，たとえば，VM/370 は1台のシステム/370 を図 9.22 に示したように多数のシステム/370 のように見えるようにする．VM/370 は，電話会社が同じ回線上に別口の多数の会話を多重化して通信するように，物理的なハードウェア資源の多重化を制御してこの離れ業をやってのける．

VMM の概念は，一度理解してしまうと，きわめて単純で論理的である．不幸にして，このシステムの概念は大部分の通常のオペレーティング・システムとは相当に異なっているので，多くの人達はこれを理解するのに困難を感じている．本節の説明に加えて，Buzen (1973)，Goldberg (1973)，Madnick (1969)，および Parmelee (1972) の報告をおすすめする．

最初，裸の計算機のインターフェースを模写するという考えは，出発点に戻ってしまうので，馬鹿げたことだと思われた．大きな相違点は，VM/370 は複



(a) 実のシステム/370　　(b) 複数の仮想システム/370

図 9.22 実計算機および仮想計算機

数の裸の計算機を作り出す効果があるということだけである．このようにすると，各利用者は自分専用の370計算機を所有しているようにみえる．したがって，各利用者は，“個人用”計算機を運転するのに好きなオペレーティング・システム（OS/360, DOS など）を選択することができる．このことは図 9.22 でオペレーティング・システム OS_1, OS_2, OS_3 で示されている．仮想計算機のインターフェースとオペレーティング・システムの間の関係をもっと図式的に Buzen（1973）が描いたのが図 9.23 である．

VM/370 はこの離れ業をどのようにして行なうのであろうか．VM/370 の利用者はどのようにしてシステムと通信すればよいのか．VM/370 の下で実行するプログラムは，通常，オペレーティング・システムであるが，物理的にはプ

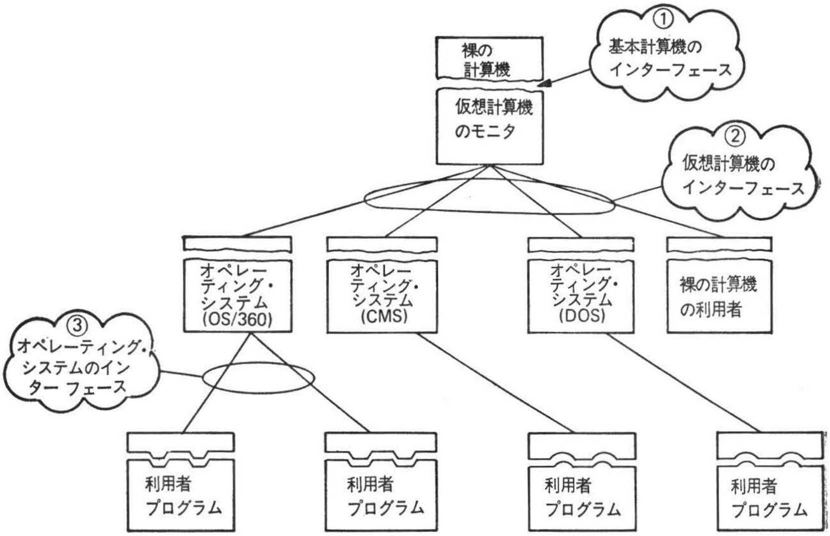


図 9.23 仮想計算機の構成

ログラム状態で実行していても，あたかもスーパバイザ状態であるかのごとくに実行する．入出力起動や記憶キーの設定などの特権命令を実行すると，割込みが発生して制御が VM/370 に移される．この割込みは，特権命令が実際に実行されたかのごとくに処理される．したがって，これらの特権命令割出しは利用者と VM/370 との間の巧妙なインターフェースである．

9.5.1.1 歴　　史

制御プログラム 40/Cambridge モニタ・システム（CP-40/CMS）の仕事は，1964年に IBM の Cambridge 科学センターにおいて開始された（Parmelee, 1972）．この計画では，単純なデマンド・ページングが可能なようにハードウェアを改造した IBM システム/360 モデル40を使用した．ソフトウェアには次の2つの部分があった：

1. 制御プログラム—40：仮想計算機のモニタ
2. Cambridge モニタ・システム：単純で使いやすい1人用のオペレーティング・システム

CP-40/CMS 計画は次のことを目的とする研究であった：

1. 各種のタイムシェアリング技法と手法の研究
2. タイムシェアリングのためのハードウェアの条件の評価
3. 構内用のタイムシェアリング・システムの組込み
4. オペレーティング・システムとハードウェアの間の動的な相互関係を観測する手法の開発

CP-40/CMS システムは1967年の始めまでに IBM の CSC で一般に使用されるようになり，最大15人の利用者にサービスを行なった．

CP-40/CMS から VM/370 へ発展するときに次の2つのことが大きな影響を与えた．第1に，第9.2節で述べた IBM システム/360のオペレーティング・システムの分芽繁殖によって，互換性や特殊な機能のために2つ以上のオペレーティング・システムを運転しなければならないという悩みが生じた．第2に，特殊なシステム/360モデル67のハードウェア（デマンド・ページングのために動的番地変換のハードウェアを備えた特別のモデル）で運転するように設計された IBM のタイムシェアリング・システム/360（TSS/360）の開発が遅れていた．この結果，モデル67のハードウェアで運転させる CP-40/CMS を開発することが1966年に開始された．1967年には，特別のモデル67のハードウェアを有する計算設備で CP-67/CMS が使用できるようになった．CP-67/CMS の開発が速かった理由の1つは，CP-67 と CMS の要素とを明確に分離したことにあったとされている．

1972年に発表された VM/370 システムは，CP-40/CMS と CP-67/CMS システムに続くものである．このシステムは，第3.6節，第3.7節および第3.8節で述べた動的番地変換を備えたものであれば任意の IBM システム/370で運転することが可能である．

9.5.1.2 使用法と原理

仮想計算機は多数の使用法と利点を有している（Buzen, 1973）：

1. 異種のオペレーティング・システムを必要とする別の利用者の同時実行たとえば，1人の利用者は新しい OS の検査を行なっており，他の人は検査のすんだ OS を利用している．
2. 変換プログラムの廃止　利用者は DOS プログラムを DOS の下で，MVT プログラムを MVT の下で実行することができ，しかも両方の

オペレーティング・システムは同じ計算機で実行することができる.

3. ソフトウェアの開発　実際の計算機と異なった構成の計算機で実行するプログラムの開発とデバッグが可能である (VM/370は, より主記憶容量の大きいものなど, 実際の370と異なった仮想的な370を作り出すことができる).
4. ネットワーク機能の検査　処理装置間通信を1つの VM モニタの下で働く複数の仮想計算機で模擬することができる.
5. プログラム動作の評価　仮想計算機のモニタでは, ある種の命令は直接裸の計算機で実行するのではなく, 解釈を行なって実行させるために命令実行に介入する必要がある. このような介入を受ける命令には入出力要求やスーパバイザのオーバヘッドなどの測定が必要であれば, 仮想計算機のモニタを変更してこれらの統計をとるようにしてから実行を行なえばよい. このような仮想計算機のモニタを使用して大量の実験データが収集されている (Hatfield, 1971 ; Morrison, 1973).
6. 信頼性　仮想計算機のモニタは普通大容量のプログラムや高度の論理的な複雑さを必要としない. このために, 総合的な検査手順を実行して全体的な信頼性と特別の秘密や安全性の保証を行なうことが容易となる.
7. 安全性と個人の秘密　独立の仮想計算機間が高度に分離されているので, 個人の秘密と安全性を護ることができる (Madnick, 1973). 基本的には, 利用者間の秘密は, オペレーティング・システムは仮想計算機で実行されているか裸の計算機で実行されているかわからないので, 保証される. したがって, 同時に存在する仮想計算機の内容をスパイしたり変更したりすることはできない.

9.5.1.3 CP と CMS の関係

VM/370 は次の2つの主要部分から成っている：

1. 制御プログラム (CP)——仮想計算機のモニタ：仮想計算機を作るために処理装置, 記憶装置, および入出力装置の多重化を行なう.
2. 会話型モニタ・システム (CMS)——単純なオペレーティング・システム：コマンド処理, 情報管理, および限定された装置管理を実行する (これらは, タイムシェアリング・システムのコンソールで操作している利用者が必要とする機能である. 第9.3節の CTSS の例を参照). CMS

は単一連続記憶割付けを行ない，多重プログラミングを行なっていないので，記憶管理と処理装置管理はきわめて簡単である．

CP と CMS は，通常，一諸に使用されるが，CMS の代わりに，OS/360, DOS, DOS/APL, OS/VS1, あるいは OS/VS2 などの他のシステム /360 や 370 のオペレーティング・システムを用いることもできる．VM/370 の設計では CP と CMS は明確に分割されている．システムをほぼ等量の2つの部分に分割することによって，設計と組込みは大層簡単になっている．さらに，CP も CMS も共に裸の計算機のインタフェースの上に構成されているので，直列的ではなく，並列的に組込みおよびデバッグが可能である．

CP-40/CMS と CP-67/CMS の CMS 成分は共に，仮想計算機で使用する前に，まず実の計算機でデバッグされた．実際，CMS の主要部分は，CP 成分が完成するよりも早く動作可能となっていた．将来のオペレーティング・システムの多くは仮想計算機の手法を利用することになろう．

9.5.1.4 システム/360とシステム/370の模擬

CP は，システム/370のハードウェアと共に，処理装置，記憶装置，入出力装置，オペレータ・コンソールおよびシステムに制御パネルを模擬する．各仮想計算機に対して CP は，仮想計算機の要素の特性と状態を登録した一組の表を管理している．これらの表は，親計算機 370 のハードウェア要素と仮想計算機の要素との対応をつけている．このようにして，たとえば IBM 2741 型通信端末装置などの鍵盤装置は，各仮想計算機のシステム制御パネルやオペレータ・コンソールを模擬するために使用される．

システム制御パネル　CP は各仮想計算機に1台の鍵盤装置（遠隔のものでも局所的なものでもよい.）を割付け，この装置にシステム制御パネルの主要な機能を写像する．システム/370のパネル上の RESET 釦はタイプされた文字列 RESET となり，これによって CP は，仮想のシステム/370を記述している表の中の状態情報を適当にリセットするように模擬の手順を開始する．同様にして，CP は，システム制御パネルのその他の機能の模擬をも行なう．

CPU　システム/370では，プログラム状態とスーパバイザ状態の区別があるので，CP は大部分の仮想計算機の命令を直接実行することができる．CPU がプログラム状態のときには，システムの状態を変更したり調べたりする特権

命令を実行しようとするとプログラム割込みが起こる．したがって，プログラム状態のときだけ仮想計算機の命令を直接実行し，特権命令の実行は割込みによってCPに制御が移されるようにすることができる．特権命令が実行されたときには，VMは次のようにして特権命令の機能的な効果を模擬する．仮想CPUを記述している表から状態を決定する——特に，状態がプロブレム状態かスーパバイザ状態かを決定する．仮想計算機がプロブレム状態であった場合は，CPは，仮想計算機に対するプログラム割込みの模擬を行なわねばならない．このためには，仮想計算機のプログラム旧PSW番地に仮想計算機のCPU状態を退避し，仮想計算機のプログラム新PSWを取出して，仮想CPUのためのCPの表の中のデータを更新することが必要である．仮想計算機がスーパバイザ状態であった場合は，CPは，この命令を解釈して模擬を行なわねばならない．たとえば，仮想計算機のSSK命令に対しては，CPはキーの値とブロック番地を決定し，応対するページが主記憶に入っている場合にはキーを指定された値に設定しなければならない．ページが主記憶に入っていない場合には，CPはキーの値をスワップ表の適当な欄の中に格納しなければならない．いずれの場合でも，CPは，仮想計算機が実行を再開する前に，仮想計算機の状態変化を反影するように表（とハードウェア）を適当に更新しておく必要がある．

入出力システム　すべての入出力が模擬される．仮想計算機が入出力命令を実行しようとすると，割込みが発生して，制御がCPに移される．これには主として次の3つの場合が存在する：

1. 入出力装置が物理的に存在してこの仮想計算機に割当てられている場合には，この入出力は直接実行される．
2. 類似の入出力装置が存在する場合には，入出力コマンドは適当に変更されてから実行される——たとえば，1台の大容量ディスクの別々の領域を用いて複数台の小容量ディスクを模擬する．
3. ある種の装置は，スプーリングと同様の技法を用いて模擬される——たとえば，仮想のカード読取り機および印刷機（第5.6節参照）．

記憶装置　デマンド・ページングによって各仮想記憶に独自の仮想記憶を作る．

9.5.2 ハードウェア

VM/370のCP要素は，動的番地変換機構を備えた標準のIBMシステム

/370のハードウェアを使用する．利用者の仮想計算機におけるプログラムで実行される非特権命令はハードウェアによって直接処理することができる（ページ割出しが発生することがある）．特権命令はプログラム割込みを起こす．この割込みは CP のソフトウェアによって特別の処理が必要である．記憶番地の特定の部分（タイマ，PSW, CAW, CSW など）および入出力装置機番はハードウェアで直接変換されるのではなく，CP によって処理される．システム/370にはモデルによっては特別の“マイクロ・コード”の機能が備わっており，これを用いることによって特殊な割込み処理は直接ハードウェアによって実行することができる．

処理装置の多重化，および仮想計算機が一部分直接ハードウェアで実行され一部分 CP のソフトウェアで模擬されていることから，厳密に時間関係を保証することはできない．これ以外の大部分のシステム/370の機能は仮想システム/370によって再現することができる．

多くの計算機が自分自身を“仮想化”しようとしなかったことは興味あることである（Goldberg, 1973）．このようなことは，“微妙な”命令（仮想計算機の状態に影響を与えたり検査したりする命令）が特権命令でないときにも起こる．たとえば1つの計算機では，プロブレム状態で実行しているときには特権命令はすべて無視される（これらは“不動作”として処理され，割込みを起こさない）．別の計算機では，スーパバイザ状態からプロブレム状態に移行する命令（システム/370 では LPSW によって実行される）は特権命令ではない．このような場合には，仮想計算機モニタは仮想計算機がどちらの状態で実行しているかを決定することができない．仮想計算機が正しくしかも能率よく動作するための計算機の設計を形式的に定義することは，研究上非常に興味のあることである

9.5.3 記 憶 管 理

CP では記憶管理にポストページングを行なうデマンド・ページングを行なっている．全般的なページング方式は，第 3.6.2.7 節で述べたのと同様の似近 LRUに基づくものである．各仮想計算機の特性を記述するために CP では表を用いている．この表には仮想計算機の記憶容量も納められている．CP は，仮想記憶に最大16Mバイトの仮想記憶を提供することができる．通常の各仮想計算機の記憶容量の 320K バイトに制限される．

CP では，システム/360の番地指定（動的番地変換を持たない）と動的番地変換を持つシステム/370の番地指定のいずれをも用いることができる．たとえ

ば，基本番地指定モードで番地変換割出しが発生すると，CP はこの番地が仮想計算機の記憶域の番地の範囲に入っているかどうかを決定しなければならない．もし範囲の外側であれば，仮想計算機に番地指定割出しが起こったことを通知しなければならない．もし内側に入っていれば，記憶ブロックを空けて(ページ置換えアルゴリズムの機能)，必要とされたページの写しの2次記憶装置の番地をスワップ表の中で求め，このページを主記憶に読込み，スワップ表に指定された値を記憶キーに設定しなければならない．これらの処理が完了すると，CP は仮想計算機の運転を再開することができる．

動的番地変換を行なう仮想のシステム/370を構成するには上記以外のソフトウェアの機構が必要である．この機構は，ページ割出しを CP あるいは仮想計算機のいずれか一方のこの割出しの処理を受持つべき方に通知しなければならない．ページ割出しは次の2つの原因で発生する：

1. 仮想計算機が自分で番地空間からページを取除いている(実の計算機で実行してもページ割出しは起こる)．
2. ページは CP の記憶管理によって取除かれている(このページ割出しは実の計算機では起こらない)．

ページングを行なう仮想計算機をページングを行なっている実の計算機で組込むためには，仮想計算機でのページの状態だけではなく，実際の状態をも管理するために余分の表が必要である．別の言葉でいうと，ページングのハードウェアの機能を模擬する必要がある．

番地変換を行なう仮想のシステム /370 (これを V370R と呼ぼう) を構成するのに CP は，親計算機 370 の DAT ハードウェアを利用して，V370R の変換ハードウェアの模擬を行なう．OS/VS1 と同様の仮想番地から“実”番地への変換を示す V370R のオペレーティング・システム内のセグメント表およびページ表は，V370R の“実”番地から親システム /370 の主記憶への写像を示すCPの表と組合せる必要がある．CPでは，この組合せを行なった表は**陰のセグメント表**および**陰のページ表**と呼ばれている(図 9.24 参照)．

これらの陰の表の使用法を説明するための短い例を示そう．CP のマップは仮想計算機の記憶装置のページ6が，親のシステム /370 の記憶装置のページ100 に入っていることを示している(図 9.24，点A参照)．仮想計算機が再配置モードで実行している V370R であれば，仮想番地を実番地に対応づける表を持っている．たとえば，ページ16はブロック6に入っていることを示している(図 9.24，点B)．

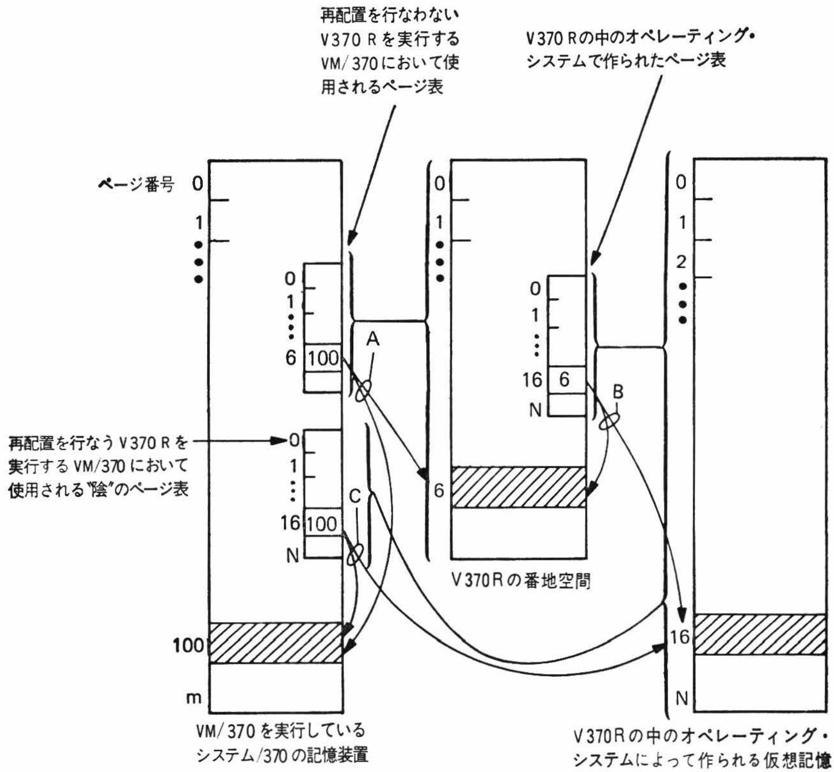


図 9.24 動的番地変換機構を備えた仮想システム/370を構成するための陰の表の使用

仮想番地から“実”の記憶装置への V370R のマップの模擬を可能にするには，CP は仮想ページ16を実ページ100に写像する陰の表を構成しなければならない（図 9.24，点C）．V370R を非再配置モードで実行するときには CP は通常のページ表を用い，V370R が再配置モードになるときには CP は陰の表を用いる．このようにして，CP は，V370R において OS/VS1，OS/VS2，および VM/370 自身を実行できるように忠実な模擬を行なっている．

記憶管理機構のもっと詳細な説明は Parmelee (1972) および VM/370 マニュアルに示されている．

9.5.4 処理装置管理

CP の処理装置管理は次の点で MULTICS (第9.4節) と類似している：

1. スラッシングを防止するために多重プログラミングするジョブ数を制限する.
2. 会話型ジョブを優先的に実行する.

CP におけるプロセス状態図は図 9.25 に示されている.

多重プログラミングで実行されているジョブは (MULTICS 流には適格ジョブと呼ばれている) Q1 および Q2 と呼ばれる 2 つの行列に分割されている. 残りのジョブは Q1 あるいは Q2 に対する保持状態として管理される. 入出力の多いジョブおよび会話型ジョブは Q1 に入れられ, CPU での計算が多いジョブは Q2 に入れられる. スラッシングを防止してシステムの利用効率をよくしておくために, Q1 および Q2 に入ることを認められるジョブの個数に制限が設けられている. Q1 および Q2 の典型的な制限数は図 9.26 に示されている.

始めてシステムに入った新しいジョブや端末の利用者からの入力を受取ったジョブは Q1 に対する保持状態に置かれる. Q1 実行可のジョブに空きが生じると Q1 保持の中の最も古いジョブが選択される. (ジョブは Q2 保持から Q2 実行可へも同様にして遷移する.) Q1実行可のジョブはラウンド・ロビンの順番で選択される. Q2 実行可のジョブは Q1 実行可のジョブが存在しないときに限って実行される. したがって, 会話型あるいは入出力の多いジョブが優先的に実行されることになる.

ジョブがコンソール入出力を待合せる必要があれば, Q1 および Q2 から取除かれる. 入力が受取られるとこのジョブは再び Q1 保持になり上述の手順を繰返す.

ジョブが Q1 あるいは Q2 から Q2 保持に置かれるのは次の場合である：

1. ジョブが Q1 に入っており,
   a. 入出力を要求せずに割当て時間 (たとえば50ミリ秒) 以上実行している場合, あるいは,
   b. コンソール入出力を要求せずに CPU 時間の合計が 400 ミリ秒になった場合.
2. ジョブが Q2 に入っており, コンソール入出力を要求せずに CPU 時間の合計が 5 秒になった場合.

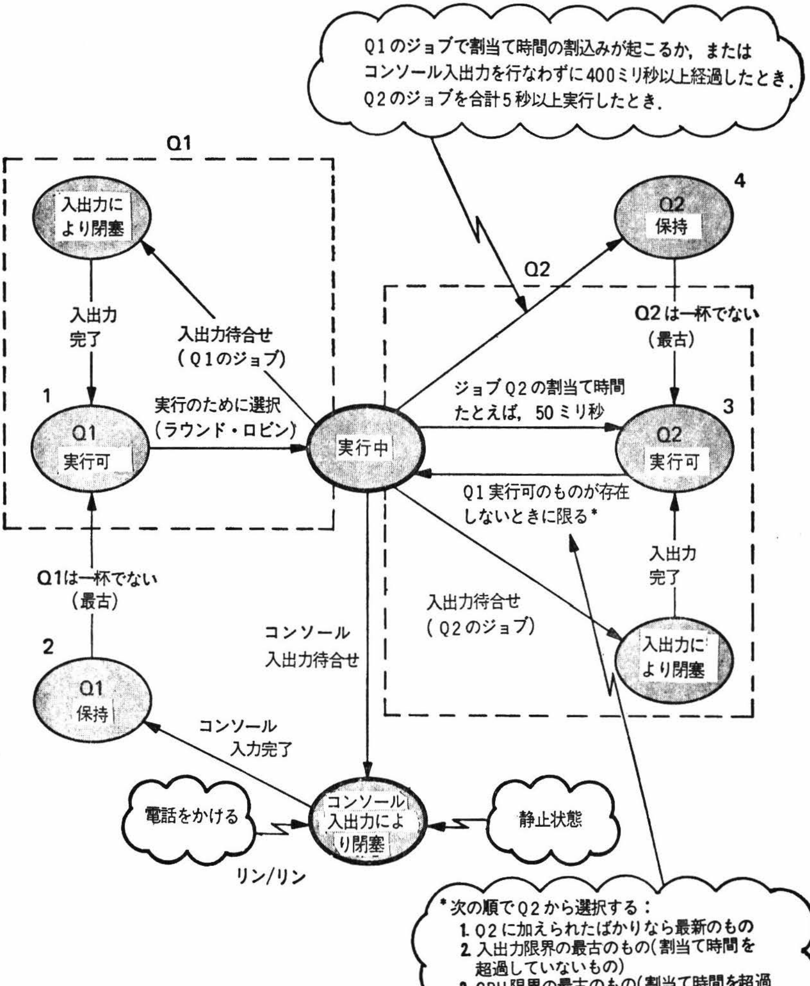


図 9.25 CP-67/CMS の要素の CP プロセス

主記憶容量(バイト)	256K	512K	768K	1024K
Q1 制限	3	6	9	12
Q2 制限	1	3	6	9

図 9.26 CP-67のジョブ/プロセス・スケジューラの典型的な行列の容量

計算機のオペレータは，システムに接続を許される利用者数の合計に対する制限を設定することができる．

9.5.5 装置管理

CPでは装置管理の3つの方法のすべてを使用している：

スプーリング——読取り機，印刷機，穿孔機の入出力に対して
専用——磁気テープ入出力に対して (CPでは他の装置でも専用使用を行なうことができる．たとえば，ディスク，印刷機)
共用——ディスク (ミニディスク)，通信制御装置 (回線) のための入出力

専用あるいは共用のためには，CPは次の手順を実行しなければならない：

1. CCWをシステム領域に複写する．
2. CCWの番地を物理番地を指すように変換する (必要ならばデータ・ページを主記憶に読込む)．
3. ページ境界にまたがるデータ領域を指定しているチャネル・コマンドを複数のデータ・チェインしたコマンドに変換する．これは，一般に仮想のページでは連続していても実の記憶装置では連続していないからである．
4. 入出力をスケジュールする．
5. 入出力が完了したとき，仮想計算機に対して入出力割込みの模擬を行なう．
6. データ・ページを解放する．

スプーリングは，第5章で述べたように，割込み駆動型で行なわれる．端末装置はただ1つの入出力CCWの鎖を用いて，部分的にスプーリングされる (特別のCPバッファに複写されている)．

装置管理は，各仮想計算機の入出力構成を記述した表を管理して，すべての

装置を管理している．これらの表は各入出力要素（仮想の装置）の存在を示すだけでなく，各要素の状態（たとえば，動作中あるいは空き）およびこれに対応している実際のハードウェア装置についても表わしている．仮想計算機がSIO 命令を実行したとき，CP はまず入出力装置の機番が仮想計算機の入出力構成で正しいこと，および，仮想の入出力経路（チャネル，制御装置，装置）を構成している要素が空いていることを決定しなければならない．次にCPは，仮想経路が動作中であると印をつけ，実のハードウェアに対して等価な入出力タスクを作らなければならない．

最も単純な例を考えると，IBM 3330 型直接アクセス記憶装置の番地 190 に対する仮想計算機の SIO 命令は実の 3330 の番地 332 に対して CP が SIO 命令を実行することになる．もちろん，実際の経路は動作中であることもあり（他の仮想計算機に対する入出力要求で要求されたチャネルが利用中のとき），このような場合には，実の経路が解放されるまで新しい入出力要求は待たされる．実の経路が空いていれば，CP は SIO を実行することができ，仮想計算機の SIO 命令に続く命令の実行を行なうことができる．入出力要求が完了したときには，CP はこのことを仮想計算機の入出力構成を記した表に記入しなければならない．特に，前には動作中であった仮想経路が解放され，割込みが起こっていることを表示しなければならない．次に，仮想 CPU で割込みが可能になっていれば，CP は，仮想計算機のチャネル状態語の更新をも含めて，割込みの効果を模擬しなければならない．

仮想計算機の入出力構成の要求に対して直接対応する装置が存在する場合には前述の入出力処理手順が実行される．直接対応する装置が存在しない場合には CP は仮想計算機の入出力構成におけるデータの流れを詳細に模擬しなければならない．たとえば，仮想計算機で2311型直接アクセス記憶装置（旧式の記憶装置）を必要とするオペレーティング・システムを実行したい場合には，この装置は3330型記憶装置を用いて模擬することができる．

9.5.6 情報管理

裸のシステム/370のハードウェアには情報管理機能が備わっていないので，VM 370 の CP 要素は情報管理機能を備えていない．情報管理は，通常，VM/370 によって構成した仮想計算機で実行されるオペレーティング・システムを適当に選ぶことによって達成される．会話型モニタ・システム（CMS）はこのようなオペレーティングシステムの１つであり，VM/370 の１つの要素として提供される．CMS 自体は，原理的には，１人用のオペレーティング・システ

ムとして裸のシステム/370で実行することができるが，特に仮想計算機の中でうまく実行できるように設計されている．

CMS は2台以上の2次記憶装置を持つシステム/370の構成（仮想あるいは実の）を必要とする．仮想計算機の場合には，これらの記憶装置は実の2314あるいは3330型装置（専用割付け）であってもよいし，あるいは，もっと普通には，通常の容量よりも小容量（たとえば，標準の200シリンダの2314ではなく10シリンダの2314）を持つ装置のように模擬している実の装置の一部分（1以上のシリンダ）である．

CMS は，OS/360 の VTOC と同様にして，各記憶装置に対して別個のファイル・ディレクトリを管理している．システム・ディスクと呼ばれる1つの記憶装置には CMS の非常駐の部分と，アセンブラ，コンパイラ，あるいはユーティリティ・プログラムといった各種のシステム・プログラムが格納されている．他の記憶装置には，図 9.27 に示すように，各利用者の恒久的なファイルが格納されている．

CP では2台以上の仮想の記憶装置を同一の実の記憶装置に写像することができる．したがって，通常，システムには実の CMS システム・ディスクは1台だけ存在していて，すべての CMS 仮想計算機は同一のディスクを共用している． CP では利用者がシステム・ディスクの内容を変更しないように保護することができる（読出し専用で書込み禁止）．

CMS のファイルは次の3要素によって規定される：

ファイル名
ファイル種別
ファイル・モード

ファイル種別はファイルの使用法を示す．たとえば，FORTRAN は FORTRAN の原始のファイル，TEXT は目的（機械語）ファイル，LISTING は，コンパイラやアセンブラの出力である．ファイル・モードはディスクの種別を示す（“S”はシステム・ディスク，“A”は利用者用ディスクである）．したがって，TEST FORTRAN A は TEST という名前の FORTRAN 原始プログラムであり，利用者用のディスクに入っている．多くの CMS のコマンドにおいてはファイル種別やファイル・モードは無指定条件があらかじめ定めてあるので省略してもよい．

図 9.28は CMS ファイル・システムで使用される記憶装置の構成を示している（幾分簡略化してある）．装置全体は800バイトのブロックに分割されてい

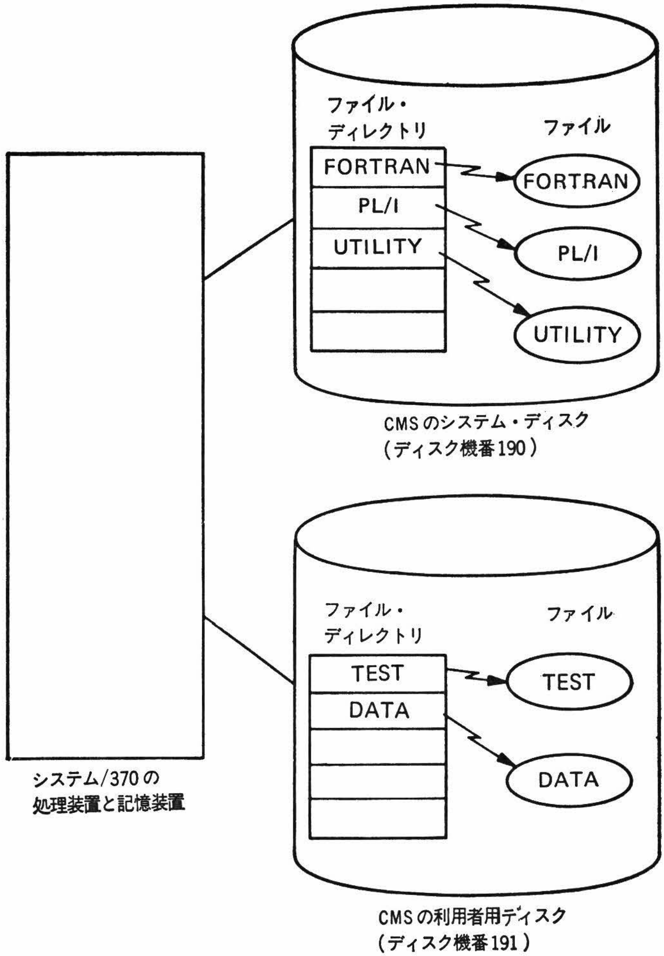


図 9.27 CMS のファイル・システム

る．幾つかのブロックはファイル・ディレクトリや領域割付けディレクトリを格納するために使用される．(図 9.28には示されていないが，マスタ・ディレクトリ・ブロックを用いて，ディレクトリの位置を示している．) 通常の動作を行なっている間に，これらのディレクトリは主記憶の中に複写される．その他のブロックは次の用途に供される：

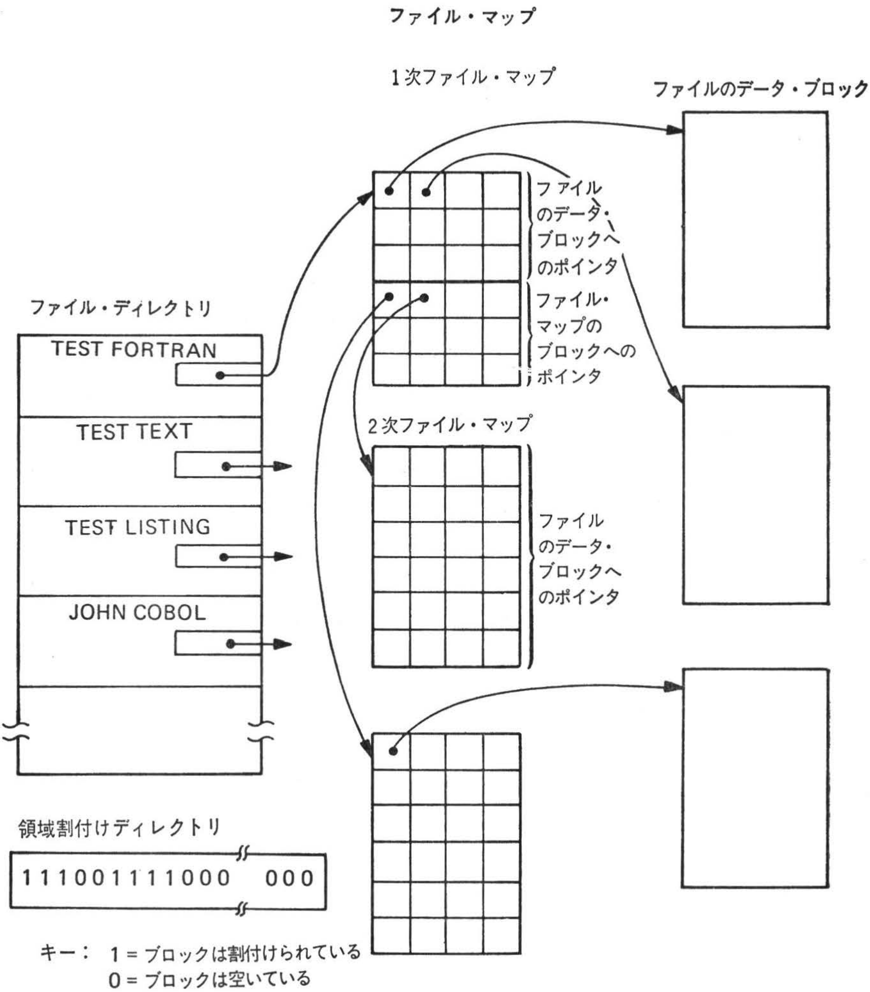


図 9.28 CMS のファイル・システムで使用されている記憶装置の構成

1. ファイル・マップのブロック
2. ファイル・データのブロック
3. 未使用ブロック

領域割付けディレクトリは，未使用のブロックを管理するために使用される**ビット・マップ**である．この装置にn個のブロックがあれば，このディレクト

りの長さはnビットである．したがって，4,000ブロックの装置容量4,000×800＝3,200,000バイト）では4,000ビット（500バイト）の領域割付けディレクトリが必要である．i番目のビットが0のときは番号iのブロックは未使用である．

ファイル・ディレクトリには，この記憶装置に入っている各ファイルに関する項目が納められている．各項目は，名前，長さ，およびその他の状態情報を示している．さらに，各項目は，このファイルの1次ファイル・マップの記憶ブロック番号を示している．1次ファイル・マップはブロック番号を示した2つの表から成っている．1つの表は最大60個のデータ・ブロックを指定し，他は最大40個の2次ファイル・マップ・ブロックを指定する．

2，3の例を用いてこれらのファイル・マップの使用法を説明しよう．80バイトのレコード250個から成る原始ファイルは20,000バイトの記憶域を必要とする（800バイトのファイル・データのブロック25個）．これら25個のデータ・ブロックの番地は1次ファイル・マップに格納される．2次ファイル・マップは必要ではない．32バイトのレコード5,000個から成っている出力リスティングのファイルは660,000バイトの記憶域を必要とする（800バイトのファイル・データのブロック825個）．最初の60個のデータ・ブロックの番地は1次ファイル・マップに格納される．残りの765個のデータ・ブロックの番地を格納するにはさらに2個の2次ファイル・マップが必要である（それぞれ400個のブロック番号を格納する）．CTSSのファイル・システム（第9.3.6節に論じた）と比較して，CMSのファイル・マップの構成によるとファイル中のレコードにより能率よく直接アクセスすることができる．

演 習 問 題[3)]

9.1 HIS 645では，セグメント表の欄の中のビットの指定によって，セグメントを64語のページか1024語のページで構成するようにすることができる．

1. 64語のページが，特に小さいセグメントに対して，ページ余りを減少させるのにいかに有効かを説明せよ．
2. システムで64語のページと1024語のページの両方を用いるとどのような問題が生じるか．

3) 本章の副読本として OS, CTSS, MULTICS, VM/370, あるいはその他のオペレーティング・システムの詳細なる参照マニュアルを使用している場合には，これらのシステムの資源管理に関してより詳細な説明をさせる問題を出すのがよい．

3. 問2の問題点を解決する方法を示せ.

9.2 CTSS ではすべての入出力は，入出力の同時処理や多重プログラミングを行なわず，直接実行された．多重プログラミングと，スワッピングおよび実行との同時処理とを行なうための技法を説明せよ．これらを変更したときの性能の変化の予想を述べよ．

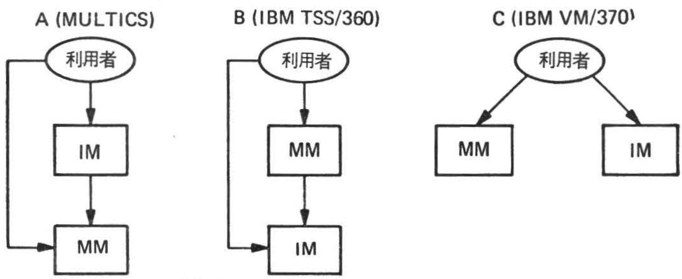
9.3 MULTICSでは，大部分の入出力（ページング以外）はシステムのバッファ領域を用いて実行される．データはシステム領域と利用者のバッファ領域との間で転送される．これに対して，VM/370 では利用者のバッファを収容しているページを主記憶中でロックをかけて，入出力を利用者のバッファとの間で直接実行する．

1. それぞれの方法の利点と欠点をあげて論じよ．
2. どちらの方法がよいか．その理由をも述べよ．

9.4 VM/370 では Q1 ジョブはつねに Q2 ジョブより優先的に処理される．

1. Q2 ジョブが全く実行されないような可能性はあるか．
2. Q2 ジョブに対して最低限のサービスを保証するように，VM/370 の処理装置管理の変更を述べよ．

9.5 ページングとセグメンテーションとを使用するシステムでは，情報管理と記憶管理の多くの特徴を組合せることが可能である．下図A，B，Cはシステムのこれらの部分の構成法を3とおり示したものである．各方法をとった理由を簡潔に述べよ．



MM = 記憶管理，IM = 情報管理

9.6 IBM 360 MVT システムは基本的には割込み駆動型である．すなわち，1人の利用者が割込みを起こすと（入出力または SVC），他のプロセスが開始される．しかし，IBMではタスクに割当て時間を指定する任意選択機

能をも用意している.

1. 各方法の長所と短所を論じよ.

2. これらの方法において，優先権によるスケジュールをどのようにして組込んでいるか.（ヒント： 主記憶中には同時には9個以上のジョブが存在することはない.）

IBM では ROLL-OUT/ROLL-IN によって最初に指定された領域を越えてジョブを動的に拡大することができる. ジョブが追加の記憶域を必要とすると，この機能は未使用の領域を求める. もし未使用の領域が求まらない場合には，閉塞されたジョブをロール・アウトして，その記憶域を要求したジョブに臨時に割当てる. ジョブがこの記憶域を解放すると，元のジョブが再びロール・インされる.

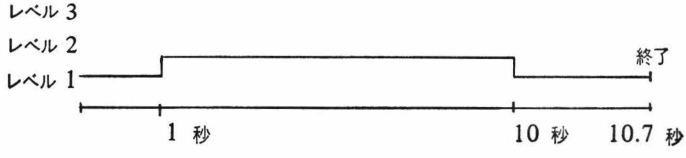
3. この機能の価値を論じよ. 時間や優先順位などを考慮せよ. この機能を使用するとどのような問題点が生じるか.

9.7 本章では CTSS のプロセス・スケジューリング・アルゴリズムを論じた（第9.3.4.2参照). 基本割当て時間を Q（CTSS では500ミリ秒)，最大静止時間を X（CTSS では1分間）とする. あるジョブが X 分間実行されなかったら，このジョブは次に高い優先順位のレベルに移される.

以下のジョブでは，$Q=100$ ミリ秒，$X=1$ 秒と仮定する.

ジョブ・コマンド	実行時間	容量
エディタ	400ミリ秒	3K
コンパイラ	5 秒	20K
APL	500ミリ秒	10K
磁気テープ・ダンプ	1 秒	2K

1. すべてのコマンドは上表の順に同時に入力されると仮定して，各ジョブに対する時間と割当て時間のレベルとの関係を示せ.



2. 次の場合にはどうなるか.

i） $Q\to\infty$

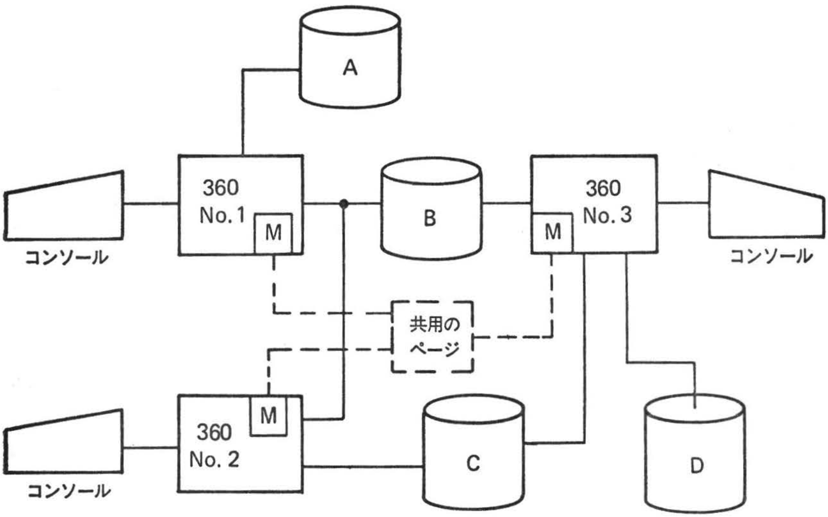
ii） $Q\to 0$

iii) $X\to\infty$

iv) $X\to 0$

3. X, Q の最適値を決定する規準は何か.

9.8 下図は IBM CP-67 タイムシェアリング・システムの論理的な構成を示したものである.



各利用者には, 専用の記憶装置 (共用していない) を有する専用の 360 が与えられているように見える. ディスク自体は 2 人以上の利用者専用の 360 間で接続されるようにできる. たとえば, ディスク B は 3 人の利用者の 360 のすべてに接続されているが, ディスク C は利用者 2 および 3 のみに接続されている. したがって, 共用のディスク装置を通じてある種の共用が可能である.

1. ディスク装置を通じて以外には物理的に共用している記憶装置が存在しないと仮定すると, ファイル共用 (たとえば, 共用ディスクのファイルを 2 人以上の利用者が読み書きすること) の機能を開発するうえで問題点があるか.

(註: 各利用者の360には別個のファイル・システムが存在する.)

本問は階層的なデータ構造と動的に制御された共用を追加するのに関する問題点を吟味する. CP-67 の 1 つの改訂版が用意されている.

利用者の 360 すべてに共通に使用される記憶域の共用ブロックが用意されて

いる．この共用ブロックは容量が4096バイトであり，前図の点線で示されている．(たとえば，4096～8192番地がすべての計算機に共通である．)

ファイルの読出し書込みが行なわれる度に，各利用者の共用ブロックにロックがかけられる．読出しロックと書込みロックとは別個に備けられる．同一のファイルを複数人の利用者が読出すことはできるが，一時には1人の利用者だけがこのファイルに書込むことができる．このファイルに対して他の利用者が引続いてアクセスしようとすると，書込みロックが外されるまでループする．通常，この待合せ時間は短かい．さらに，共用ページ全体に対して1つのロックがあり，360の特殊命令である判定および設定命令によって設定される．

2. ロックにはどのような情報を含めなければならないか．

3. 読出しロックと書込みロックとを区別しなければならない理由は何か．特に，読出しロックを必要とする理由は何か．

4. 共用のページにロックが備けられる理由は何か．このロックを用いないとどのような問題が起こるかを述べよ．

5. ロックは個々の360計算機で管理されているので，多数の問題が起こり得る．これらの問題点を述べよ．(**ヒント：** 各利用者は，プロセスを停止させて一時的に計算機から離れるためのSTOPボタンを持っている．) 共用されているのは4096バイトのブロック1つだけである．これによって何が起こるか．

6. 問5の問題点に対する解法決，あるいは，動的な共用を行なうためのより優れた設計を提案せよ．たとえば，1人の利用者が共用ページや特定のファイルをアクセスしようとして待合せているとき何をしたらよいかを決定してみよ．

9.9 HTS社(つまらないタイムシェアリング会社の意)は，約50人の遠隔利用者(ジョブ)を同時に処理する小さなタイムシェアリング会社である．HTSでは，標準の幸運I型実時間多重プログラミング・タイムシェアリング万能オペレティングシステム(ページングとセグメンテーションを行なっている)を使用している．主任販売員としてFrank Fasttalker(早口のフランクの意)を雇ってからは，HTSは100人の利用者があり，適当な仕事量を維持するのに問題が生じている．HTSの計算機部のHarry Moredoughは仕事量を改善するのに次の4つの変更を提案している：

a．主記憶を増す．

b．新型で高速の2型CPUを購入する．

c．旧式の象A型ドラムを新しくて高速のカバB型ドラムに変更する．

d．情報管理機能で改良された幸運Ⅱ型オペレーティング・システムに切換える．

これらの提案のおのおのについてジョブ混合体，CPU 限界の条件など，

1. 変更が大きな影響を与えるもの
2. 変更してもほとんど改善されないもの

について述べよ．

9.10 読者は OS/360，MULTICS，および VM/370 の構造をよく理解している．標準の 360 では再配置，ページング，あるいはセグメンテーションのハードウェアを備えていなかった．したがって，OS/360 では分割割付け方式を採用している．）

読者は貸し計算機会社の主任システム・プログラマである．IBMの新型の370が開発されたので，会社の倉庫には360（何台かの360/67を含む）が一杯入っている．360/67はよく出来た計算機であるので，すべての 360 に 360/67のページング・セグメンテーションの機能を含めるようにすることが提案されている．

大部分の利用者は現在のところ OS/360 を使用しており，それを使用し続けることを望んでいる．主任プログラマに67の機能を利用するように OS/360 を順次変更することが要求されている．コーディングやデバッグは1人で行なわなければならないので変更は最小にすべきである（もちろん，1日24時間働いてもよい）．

1. どのような利用者の場合には，この新しいハードウェアやオペレーティング・システムが能率向上に役立たないか．
2. どのような利用者の場合には，この新しいハードウェアやオペレーティング・システムが能率向上に役立つか．
3. 記憶管理はどのようにするか．特に，現在の OS/360 の記憶管理にどのような変更を行なうか．
4. ページングの使用は装置管理に対してどのような問題を生じるか．優れた入出力性能を達成するために，これらの問題点をどのように解決するか．
5. 現在の OS/360 で高順位ジョブを入れるために主記憶から低順位ジョブを取除くと（ロール・アウト）どのような問題があるか．ページング

／セグメンテーションのハードウェアはこの問題を解決するのに役に立つか.

9.11

1. OS, CTSS, MULTICS, および VM/370 よりも優れているオペレーティング・システムを定義して説明せよ.
2. 完全な流れ図を作成せよ.
3. このオペレーティング・システムを 360 の機械語あるいは PL/I で組込め.

第10章

註釈つき参考文献

10.1 註釈つき参考文献の相互参照[1]

基本的な原理
Altucher ほか, 1966
Donovan, 1972
Kapur, 1970
Knuth, 1968
Martin, 1971
Morris, 1968
Pankhurst, 1968
Presser, White, 1972
Price, 1971
Rosin, 1969

デバッギング・エイド
Josephs, 1969
Kulsrud, 1969
Van Horn, 1967

モニタ，スーパバイザ
Gaines, 1971
Huxtable, Warwick, 1967
Reiter, 1967

ファイル・システム（情報管理）
Bensoussan, 1969
Chapin, 1969
Katzan, 1971
Madnick, 1969

ハードウェア
Ahearn, Dishon, Snively, 1972
Amdahl, Amdahl, 1967
Cashman, 1972
Femling, 1971

高水準言語
Boehm, 1973
Corbató, 1969
Donovan, 1972

---

1. 読者の便宜のために，参考文献を主題ごとに類別しておいた．註釈つきの参考文献はこの後に続けて載せてある．

歴 史

Rosen, 1969
Rosin, 1969

ジョブ制御言語

Brawn, 1970
Cadow, 1970

言語処理プログラム

Donovan, 1972
Freiburghouse, 1969
Mclure, 1972

記憶管理（設計の原理）

Abate, Dabner, 1969
Belady, Knehner, 1969
Coffman, 1969
Considine, Weis, 1969
Denning, 1968
Denning, 1967
Dennis, 1965
Femling, 1971
Fields, 1971
Fotheringham, 1961
Funchel, Heller, 1967
Johnson, 1973
Katzan, 1971
Liptay, 1968
Madnick, 1973
Mattson ほか, 1970
Meade, 1970
Randell, Kuehner, 1967
Reiter, 1967
Risko, 1968
Ross, 1967
Teorey, 1972
Teorey, Pinkerton, 1971
Vareha, Rutledge, Gold, 1969
Wilkes, 1965

マイクロプログラミング

Hussen, 1970
Rosin, 1969
Wilkes, 1969

モデル作成

Abate, Dubner, 1969
Arden, Bottner, 1969
Belady, 1966
Buzen, 1971
Buzen, 1973
Coffman, 1969
Coffman, Ryan, 1971
Denning, 1969
Denning, 1967
Denning, Schwartz, 1972
Drake, 1967
Estrin, Muntz, Uzgalis, 1972
Feller, 1968
Fuchi, Tanaka, Manago, Yuba, 1969
Goldberg, 1973
Katz, 1966
Kay, 1972
Kleinrock, 1970
MacDougal, 1970
Madnick, 1968
McKinney, 1969

Nielson, 1967
Oppenheimer, Weizer, 1967
Schwartz, 1973
Sekino, 1972
Teorey, 1972
Teorey, Pinkerton, 1971
Winograd, Morganstein, Herman, 1971

MULTICS

Bensoussan, Clingen, Daley, 1969
Corbató, 1969
Corbató, Vyssotosky, 1965
Corbató, Clingen, Saltzer, 1972
Daley, Dennis, 1968
Feieritag, Organick, 1971
Freiburghouse, 1969
Organick, 1972
Saltzer, Gintell, 1969
Schroeder, 1971
Sekino, 1972

多重プログラミング

Arden, Bottner, 1969
Browne, Lan, Baskett, 1972
Buzen, 1971
Hellerman, Smith, 1970
Lehman, Rosenfeld, 1968
MacDougal, 1970
Schwartz, 1973

オペレーティング・システムとその原理

Betourne, Boulenger, Ferrie, Kaiser, Krakowiak, Mossiere, 1970
Conti, Gibson, Pittowsky, 1968
Crisman, 1965
Denning, 1971
Dennis, Van Horn, 1966
Dijkstra, 1968
Donovan, 1972
Enwin, Jensen, 1970
Gaines, 1971
Hansen, 1970
Ho-Nien, Peck, Pollard, 1969
Jones, 1969
Katzan, 1970
Liskov, 1972
McFarland, 1970
Needham, Hartley, 1969
Saltzer, 1966
Sevick ほか, 1972
Watson, 1970

ページング

Bard, 1973
Belady, Nelson, Shedler, 1969
Brawn, Gustavson, 1968
Coffman, Varian, 1968
Corbató, 1969
Denning, 1969
Denning, 1968
Denning, Schwartz, 1972
Gustavson, 1968
Hatfield, 1972

Madnick, 1973
Oppenheimer, Weizer, 1967
Randell, Kuehner, 1968
Schroeder, 1971
Schwartz, 1973
Seligman, 1968
Spirn, Denning, 1972
Vareha, Rutledge, Gold, 1969
Winograd, Morganstein, Herman, 1971

性能の評価と測定

Abate, Dubner, 1969
Arden, Bottner, 1969
Calingaert, 1967
Cantrell, 1968
Hatfield, 1972
Karush, 1969
Lehman, Rosenfeld, 1968
Lucas, 1971
Saltzer, Gintell, 1969
Schroeder, 1971
Wulf, 1969

プロセスとプロセス間通信

Bernstein, Deflefsen, Kerr, 1969
Dijkstra, 1968
Dijkstra, 1968
Gaines, 1971
Habermann, 1972
Horning, Randell, 1973
Spier, Organick, 1969
Varney, 1971

保　護

Comber, 1969
Conway, Maxwell, Morgan, 1972
Graham, 1967
Graham, Denning, 1972
Hoffman, 1969
Lampson, 1971
Lampson, 1969
Madnick, Donovan, 1973
Molho, 1970
Needham, 1972
Scherf, 1973
Schroeder, Saltzer, 1972
Skatrud, 1969
Weissman, 1969

定理の証明

Dijkstra, 1965
Fontao, 1971
Habermann, 1971
Kahn, 1971

スケジューリング

Abell, Rosen, Wagner, 1970
Baskett, 1971
Browne, Lan, Basket, 1972
Buzen, 1971
Coffman, Kleinrock, 1968
Hansen, 1971
Kleinrock, 1970
Lampson, 1968
Oppenheimer, Weizer, 1967
Rodriguez-Rosell, 1971

Teorey, 1972
Teorey, Pinkerton, 1971
Varney, 1971

構造的プログラミングと高信頼度ソフトウェア
Dahl, Dijkstra, Hoare, 1972
Liskov, 1972
Mealy ほか, 1968

システムのすくみ
Bensoussan, 1969
Coffman, Elphick, Shoshani, 1971
Dijkstra, 1968
Easton, 1971
Fontao, 1971
Habermann, 1969
Havender, 1968
Holt, 1972
Murphey, 1968
Saltzer, 1966

システム・プログラム
Donovan, 1972
Neumann, 1969
Presser, White, 1972

タイムシェアリング
Considine, Weis, 1969
Corbató, 1962
Earl, Bugly, 1969
Foster, 1971
Hargraves, 1969
Karush, 1969
Kleinrock, 1970
Lichtenberger, 1965
Linde, Weissman, Fox, 1969
Madnick, 1966
McKinney, 1969
Motobayashi, 1969
Reiter, 1967
Scherr, 1965
Schwartz, 1964
Watson, 1970
Weissman, 1969
Wilkes, 1968
Winogrand, Morganstein, Herman, 1971

仮想記憶
Belady, 1966
Bensoussan, Clingen, Daley, 1969
Coffman, Ryan, 1971
Daley, Dennis, 1968
Denning, 1970
Hatfield, 1971
Madnick, 1972
Morrison, 1973
Parmelee ほか, 1972
Sayer, 1969
Winograd, Morganstein, Herman, 1971

10.2 註釈つき参考文献[1]

Abate, J. and H. Dubner: "Optimizing the performance of a Drum-Like Storage," *IEEE TC*, vol. C-18, no. 11, pp. 992-996, Nov. 1969.

本論文は，トラックごとに1つのヘッドを持ち，回転待ちを最小とするようなハードウェアを用いて行列処理を行なう2次記憶装置を解析している．記憶サブシステムにおける行列の中での処理と待合せ時間の関係が要約されている．考察の結果として，性能の予想値が図示されている．

Abell, V. A., S. Rosen, and R. E. Wagner: "Scheduling in a General Purpose Operating System," *Proceedings, AFIPS, 1970 FJCC*, vol. 37, pp. 89-96.

CDC 6000シリーズの MACE オペレーティング・システムについて，ジョブ・スケジューリングとジョブ管理技法が論じられている．スケジューリングの相互関係が発表されている．

Ahearn, G. R., Y. Dishon, and R. N. Snively: "Design Innovations of the IBM 3830 and 2835 Storage Control Units," *IBM Journal of Research and Development*, vol. 16, no. 1, pp. 11-18, Jan. 1972.

本報告は，高性能で融通性のある大記憶装置の制御装置を，マイクロ・コンピュータを応用して構成する方法を述べている．

Altucher et al. (The Computer Usage Company): *Programming the IBM System/360*, John Wiley & Sons, Inc., New York, 1966.

基本的なシステム/360のアセンブラ語が示されている．プログラミングの出発点である．

Amdahl, G. M. and L. D. Amdahl: "Fourth-Generation Hardware," *Data-*

---

1. 略号は次のとおりである：

AFIPS	American Federation of Information Processing Societies
CACM	Communications of the Association for Computing Machinery
FJCC	Fall Joint Computer Conference
IEEE TC	Institute of Electrical and Electronic Engineers, Transactions on Computers
IFIP	International Federation of Information Processing
JACM	Journal of the Association for Computing Machinery
SJCC	Spring Joint Computer Conference

mation, pp. 25-26, Jan. 1967.

Arden, B. W. and Bottner: "Measurement and Performance of a Multiprogramming System," *Second ACM Symposium on Operating Systems Principles*, Princeton University, pp. 130-146, Oct. 1969.

実用的な性能評価法を提案している．Michigan タイムシェアリング・システム (MTS) に関する詳細な測定結果がデータの解析に先立って示されている（測定技法についても論じられている）．この測定は，形式的でしかも実際のシステムのモデルに適合するデータを抽出するように考慮されている．理想的なシステム応答時間を求めるために，中央の窓口を含むモデルが用いられている．能率の評価のための負荷の尺度としては，理想的な応答時間と実際の応答時間の比を用いるのがよいという結論が示されている．

Bard, Y.: "A Characterization of Program Paging in a Time Sharing Environment," *Cambridge Scientific Center*, G320-2083, Feb. 1973.

静的なパラソル曲線を用いたプログラム動作（固定の記憶域の区画におけるページング）の解析を調査して，この曲線があまり正確でないという結論を得ている．プログラムの常駐容量やページング頻度は時間と負荷に関係する量であるので，より正確なモデルとしては，平均的な常駐容量とページング頻度の関係を示す動的なパラソル曲線（プログラムの軌跡から求められる）で規定されるプログラム動作のモデルが必要である．

Baskett, F.: "The Dependence of Computer System Queues upon Processing Time Distribution and Central Processor Scheduling," *Third ACM Symposiumoon Operating Systems Principles*, Stanford University, pp. 109-113, Oct. 1971.

ラウンド・ロビン・スケジューリングの性能の上下限を求めるために段階的な手法が用いられている．処理装置共用の中央窓口と呼ばれるモデルが使用されている．非指数的なサービス分布が用いられている．実際のデータによってこのモデルがよく合致することが検証されている．

Belady, L. A.: "A Study of Replacement Algorithms for a Virtual Storage Computer," *IBM Systems Journal*, vol. 5, no. 5, pp. 78-101, 1966.

仮想記憶の動的な性質――ブロック置換え機構――を研究するために，仮想記憶計算機が構成された理由が紹介されている．典型的なアルゴリズムを研究するための確率論的なモデルはきわめて広範囲にわたっており，特別の情報を必要としないものから，未来の参照パターンの完全な知識を用いて置換えを管

理する最適法（MIN）と呼ばれるものまである．置換えの統計量をプログラム・モニタを用いて集計し，置換えアルゴリズムの入力として用いている．次にこの置換えアルゴリズムを用いて結果を予想している．アルゴリズムは詳細に研究され，比較されている．

Belady, L. A. and C. J. Knehner: "Dynamic Space Sharing in Computer Systems," *CACM*, vol. 12, no. 5, pp. 282-288, May 1969.
各種のページング・アルゴリズムが示されている．

Belady, L. A., R. A. Nelson, and G. S. Shedler: "An Anomaly in Space-Time Characteristics of Certain Programs Running in a Paging Environment," *CACM*, vol. 12, no. 6, pp. 349-353, June 1969.
FIFO ページ置換えアルゴリズムを1つの例として用い，プログラムで使用できるページ数の関数として，プログラムのページ割出し回数の興味ある関係を示している．

Bensoussan, A.: "Overview of the Locking Strategy in the File System," *Multics Systems Programmers Manual*, Section BG. 19. 00, pp. 1-10, Nov. 1969.
リング0での並列処理において，データが共用されて値が変更されようとするとき，プロセスを同期させる手法を論じている．このためには，使用される各データ・セット（ディレクトリ，ディレクトリ・ブランチなど）にロックが必要である．同期と共用を処理するためのアルゴリズムには逆計算と呼ばれる復旧操作も含まれている．

Bensoussan, A., C. T. Clingen, and R. C. Daley: "The Multics Virtual Memory," *Second ACM Symposium on Operating Systems Principles*, Princeton University, pp. 30-42, Oct. 1969.
情報の共用は，"独立の属性とアクセス権を持ち共用可能なもの（セグメント）"をセグメンテーションすることによって自動的に行なわれる．MULTICSにおける組込みの設計に関する考察と，セグメンテーションによる番地空間を構成するための仮想記憶機構の必要性が示されている．ソフトウェアとハードウェアの相互関係についての議論も含まれている．

Bernstein, A. J., G. D. Deflefsen, and R. H. Kerr: "Process Control nad Communication," *Second ACM Symposium on Operating Systems Principles*, Princeton University, pp. 60-66, 1969.

"EXEC" は GE 600 型計算機のオペレーティング・システムである．本報告には，面白いプロセス構造を持つ多目的モニタが述べられている．プロセスは，論理セグメントと呼ばれ，ベース番地レジスタによって再配置され保護される最大4個の部分から構成されている．組込まれた通信の方法と応用例が示されている．

Betourne, C., J. Boulenger, J. Ferrie, C. Kaiser, S. Krakowiak, and J. Mossiere: "Process Management and Resource Sharing in the Multiaccess System ESOPE," *CACM*, vol. 13, no. 12, pp. 727-733, Dec. 1970.

プロセスを基本にした階層構造の計算機オペレーティング・システムの設計原理が示されている．プロセス管理，仮想記憶の動作，ファイル・システムの構成，および利用者管理の間の関係を説明するために，組込みの詳細が最初に述べられている．本報告書には，単なるさわりだけでなく詳細なる議論が行なわれている．

Boehm, Barry W.: "Software and Its Impact: A Quantitative Assessment," *Datamation*, pp. 48-59, May 1973.

本報告書は空軍システム・コマンドのための研究結果を要約している．これにはソフトウェア開発の経費と経費に関係する要因を解明している．

Brawn, B. S. and F. G. Gustavson: "Program Behavior in a Paging Environment," *Proceedings, AFIPS, 1968 FJCC*, vol. 33, pt. II, pp. 1019-1032.

デマンド・ページングの仮想記憶計算機ではプログラムの形によってプログラムの性能が影響される．本報告は，単一および多重プログラミングのジョブ実行に対して，主記憶容量，ページ置換えアルゴリズム，およびプログラムの形／プログラムの局所性について経験的な試験結果を示している．物理的なシステムや設計における方針決定——ページ置換えアルゴリズムの選択——よりもプログラムの型がより重要であることが観察されている．

Brown, G. D.: *System/360: Job: Control Language*, John Wiley & Sons, Inc., New York, 1970.

本書は，IBM システム/360 JCL およびシステムに関連した概念を簡潔に概説したものである．

Browne, J. C., J. Lan, and F. Baskett: "The Interaction of Multiprogramming Job Scheduling and CPU Scheduling," *Proceedings, AFIPS, 1972, FJCC*, vol. 41, pt. I, pp. 13-22.

CDC 6600システムのシステム・シミュレーションを行なって，ジョブ・スケジューリングとCPU スケジューリング・アルゴリズムの，システム全般の性能への関連を算定している．このシミュレーションは幾つかの興味ある相互関係にも触れている．

The Burroughs Corporation: *The Descriptor—A Definition of the B5000 Information Processing System,* The Burroughs Corporation, Detroit, 1961.

この初期の報告書は，"ディスクリプタ" とセグメンテーションによる番地空間の構成原理を紹介している．

Buzen, J. P.: "Computational Algorithms for Closed Queuing Networks with Exponential Servers," *CACM*, vol. 16, no. 9, Sept. 1973.

本報告書は中央窓口のモデルと，多くの行列網のモデルの解析に有用なアルゴリズムを簡単に述べている．

Buzen, J. P.: "Queuing Network Models of Multiprogramming," doctoral dissertation, Harvard University, 1971.

Buzen はスケジューリングの原理に関する過去の研究を詳細に調査して，総合的な文献リストをあげている．中央窓口のモデルと呼ばれる多重プログラミングの原理的なモデルを示し，このモデルに基づいて性能評価の理論を展開している．解析的なモデルの解を迅速に効率よく求める新しい計算法を示している．この領域における研究に興味を持つ人にとっては価値ある出発点である．

Buzen, J. P. and U. O. Gagliardi: "The Evolution of virtual Machine Architecture," *National Computer Conference*, pp. 291-229, 1973.

仮想計算機の構成に関する歴史的な発展と動機，および未解決の問題点が示されている．

Cadow, H. W.: *OS/360: Job Control Language*, Prentice-Hall, Inc, Englewood Cliffs, N. J., 1970.

システムに関連する概念の概説が行なわれている．

Calingaert, P.: "System Performance Evaluation: Survey and Ap-

praisal," *CACM*, vol. 10, no. 1, pp. 12-18, Jan. 1967.
仕事量，返却時間，およびアベイラビリティなどの性能の測定について定義を行なっている．命令の混合体や核の使用法が示されている．解析，模擬，および合成の役割りが論じられている．

Cantrell, H. N. and A. L. Ellison: "Multiprogramming System Performance Measurement and Analysis, *Proceedings, AFIPS, 1968, SJCC*, vol. 32, pp. 213-221.
優れた解析が示されている．

Chapin, N.: "Common File Organization Techniques Compared," *Proceedings, AFIPS, 1969, FJCC*, vol. 35, pp. 413-422.
ファイル編成，ファイル構造，およびデータ管理という幾つかの基本的な定義が示されている．順編成，および直接編成の原理を比較している．詳細でしかも要領よくまとめた情報が示されている．各種のファイルのアクセス法を分類するために，時間——流れ図の興味ある手法が用いられている．

Codd, E. F.: "A Relational Model of Data for Large Shared Data Banks," *CACM*, vol. 13, no. 6, pp. 377-387, June 1970.
将来大規模のデータ・バンクを利用する人は，計算機の中でのデータの編成法（内部表現）を知らなくてもよいようにすべきである．端末の利用者や大部分の応用プログラムは，データの内部表現や外部表現が部分的に変更されても影響を受けてはならない．データ表現の変更が問合せ，更新，および報告と記録された情報の型の自然な成長によって必要となる．

樹構造のファイルでは不十分であることが論じられている．n 項関係に基づくモデル，データ・ベースの関係の一般形，および汎用のデータ用言語の概念が紹介されている．第2節では，論理的な関係に対する処理（論理的な推論）が論じられ，利用者のモデルにおける冗長性と一貫性の問題に適用している．

Coffman, E. G., Jr.: "Analysis of a Drum Input/Output Queue under Scheduled Operation in a Paged Computer System," *JACM*, vol. 16, no. 1, pp. 73-90, Jan. 1969.

ページ入出力のスケジューリングは，効率のよい仮想記憶の設計に重要である．本報告書では，これらのシステムを目的として，回転するドラムに対するモデルが詳細に検討されている．この解析法によってドラムの利用率やその他のよく参照される統計量を求めることができる（たとえば，Sekino の論文は，

２次記憶の動作のモデルを作るのに，本報告で求められる利用率のパラメータを用いている）．よく研究されている数学的なモデルの例である．

Coffman, E. G., Jr., M. J. Elphick, and A. Shoshani: "System Deadlocks," *ACM Computing Surveys*, vol. 3, no. 2, pp. 67-78, June 1971.

短かい例を引いてすくみの説明を始め，すくみの起こる必要条件を求めるために必要な定義を行なっている．状態図がすくみを検出するための有効な手段であることが示されている．次に，検出と回復の技法が検討されている．本報告には，資源要求の情報を用いてすくみを防止する方法も示されている（完全な情報が必要な場合と完全な要求が不明なときの場合について示されている）．

本報告は，現在の技法の最も顕著な特徴を概説して，理解／読解を容易にしている．

Coffman, E. G., Jr. and L. C. Varian: "Further Experimental Data on the Behavior of Programs in a Paging Envioronment," *CACM*, vol. 11, no. 7, pp. 471-474, July 1968.

プログラムのページング特性に関して経験的なデータを収集するために，デマンド・ページングの記憶管理のもとでのプログラムの動的な動作を調べている．データは例題プログラムをインタプリーティブに実行することによって集められている．関連した統計量（ページ割出しの起こるまでの命令数）が示されている．これらの統計は，別のページ置換えアルゴリズムと記憶容量の効果を検討するのに有用である．

Coffman, E. G., Jr. and T. A. Ryan: "A Study of Storage Partitioning Using Mathematical Model of Locality," *Third ACM Symposium on Operating Systems Principles*, Stanford University, pp. 122-129, Oct. 1971.

本報告では，仮想記憶の管理に対して，２つの基本的な方式――固定の区画と動的な区画――に議論を集中している．これらの方式を検討するために局所性の確率モデルを提案している．これはワーキング・セットの考えに基づく定常的なガウス過程である．ワーキング・セットが動的に変化するときの経費を計算したりすることなど，動的な方式の利点が明確に示されている．

Coffman, E. G., Jr. and L. Kleinrock: "Computer Scheduling Methods and Their Counter Measures," *Proceedings, AFIPS, 1968, SJCC*, vol. 32 pp. 11-21.

本報告は，利用可能な優先順位スケジューリングの手法を示し，応答問題や

資源利用可能率などの性能の規準を基に，これらの手法の長所短所を論じている．また，高い利用率を達成するためのサービスを決定する手法が，能率のよい行列処理の機構の設計と関連して示されている．本報告は，スケジューリングの尺度に関して優れた参考文献である．この報告では数学的な取扱いを文献に譲って，質的な関係の議論を行なっているので読解が容易である．

Comber, E. V.: "Management of Confidential Information," *Proceedings, AFIPS, 1969, FJCC*, 35, pp. 135-143.

個人の秘密に関する哲学および倫理の問題を質量な面で解析する本手法は，完全な保護を行なうシステムの基本的な条件を示しているだけではなく，個人の秘密というものの基本的で段階的な定義を行なおうとする人にとって優れた判定基準となるという点からも価値がある．本報告は，経験豊かなシステム・アナリストにとって，述べられた規準やシステムのサービスの程度まで読取ることができるので，各レベルの研究者にとって参照することができる．

Comfort, W. T.: "A Computing System Design for User Services," *Proceedings, AFIPS, 1965, FJCC*, vol. 27, pt. 1, pp. 619-628.

Considine, J. P. and A. H. Weis: "Establishment and Maintenance of a Storage Hierarchy for an On-Line Data Base under TSS/360," *Proceedings, AFIPS, 1969, FJCC*, vol. 35, vol. pp. 433-440.

記憶容量の限定されたオンライン参照のデータベース管理は，動的な記憶ボリューム階層を構成することによって可能となる．本報告は，TSS/360などのシステムを制御するために組込まれたコマンドについて述べている．このコマンドは，オンライン・データベース・システムの動作に適している．

Conti, C. J., D. H. Gibson, and S. H. Pittowsky: "Structural Aspects of the System/360 Model 85: I. General Organization," *IBM Systems Journal*, vol. 7, no. 1, pp. 2-14, 1968.

初期のバッファ（キャッシュ）記憶システムが述べられている．

Conway, R. W. Maxwell, and H. Morgan: "Selective Security Capabilities in ASAP—A File Management System," *Proceedings, AFIPS, 1972, SJCC*, vol. 40, pp. 1181-1186.

本報告は，参照を認めていない利用者が勝手に情報を参照しないようにする予防策を説明している．ファイル管理システムの発展に関する短い報告書である．より一般的なシステムの保護の問題には触れていないが，保護の問題の重

要性が示されている.

Corbató, F. J.: A paging Experiment with the MULTICS System," in Feshbach and Ingard (eds.), *In Honor of Philip M. Morse*, M. I. T. Press, Cambridge, Mass., 1969, pp. 217-228.

MULTICS のページング・アルゴリズムが詳細に述べられていて，このようなシステム機能の組込みの問題点を明らかにしている. MULTICS のアルゴリズムは，1つの極端が FIFO で他の極端が LRU であるような特性を持っている．効率の特性の実験をも行なっている．

Corbató, F. J.: "PL/I as a Tool for System Programming, "*Datamation*, pp. 68, 73-76, May 1969.

本報告は，大規模な計算機システムの開発のために PL/I を組込んで使用することの当否を卒直に述べている．MULTICS に関して，この言語の長所と短所とが示されている．全般的な結論としては，PL/I は他の高水準言語には備わっていない特徴を有しており，開発のために必要な機能を提供することが可能であったと述べている．

Corbató, F. J., C. T. Clingen, and J. H. Saltzer: "Multics—The First Seven Years," *Proceeding, AFIPS, 1972, SJCC*, vol. 40, pp. 571-584.

設計の目標，背景となった発展の経過，および MULTICS の一般的な記述が読みやすく述べられている．MULTICS の成功をもたらした技術的な問題点については参考文献にまかせて，情報処理ユーティリティの像を示そうということに焦点が合されている．MULTICS の経験を基にした記述はシステム設計者に興味がある．

Corbató, F. J. et al.: "An Experimental Time-Sharing System," *Proceedings, AFIPS, 1962, SJCC*, vol. 21, pp. 335-344.

CTSS タイムシェアリング・システムについて論じている．

Corbató, J. F. and V. A. Vyssotosky: "Introduction and Overview of the MULTICS System," *Proceedings, AFIPS, 1965, FJCC*, vol. 27, pp. 185-196.

当初の MULTICS の設計目標と，システムを組込むハードウェアに関する議論が概説されている．

Crisman, P. A.: *The Compatible Time-Sharing System; A Programmer's Guide*, 2d ed., M. I. T. Press, Cambridge, Mass., 1965.

Dahl, O. J., E. W. Dijkstra, and C. A. Hoare: *Structured Programming*, Academic Press, London and New York, 1972.

全般的な主題は，プログラムの設計とプログラミングにおける見落しや誤りを防止するために利用できる原理を解析することである．第1章は，構造的プログラミングに関する Dijkstra の論文であり，上から下へのプログラミング法について述べている．第2章は，データ構造に関する Hoare の論文である．これは，データ構造とその組込みおよび使用の設計に対する Dijkstra の原理の応用である．第3章は，Dahl と Hoare の階層的プログラム構造を述べている．これは，初めの2つの章の合成であり，データの設計と，1967年の Simula の構成法を用いたプログラムの設計との間の密接な理論的および実際的な関係を詳しく説明している．

Daley, R. C. and J. B. Dennis: "Virtual Memory, Processes, Sharing in MULTICS," *CACM*, vol 11, no. 5, p. 306, May 1968.

MULTICS で使用されているプロセスと番地空間の概念が述べられている．MULTICS の番地指定機構がそれに続き，ダイナミック・リンキングが説明されている．プログラミングのためにどのようなシステムの機能が利用できるかを詳しく述べている．

Denning, P. J.: "Effects of Scheduling on File Memory Operations," *Proceedings*, AFIPS, 1967, *SJCC*, vol. 30, pp. 9-18.

本報告の目的は，ファイル入出力の仕事量（単位時間にサービスされる平均要求数）を最大にしようとする議論の背景として，ファイル・システムの構成を概説することである．ディスクとドラムのモデルをシステムのモデルに取入れて，これらの装置に対する各種の方針を検討している．ドラム (FCFS, SATS)，ディスク (SSTF, SATF, FLFS).

Denning, P. J.: "Equipment Configuration in Balanced Computer Systems," *IEEE TC*, vol. C-18, no. 11, pp. 1008-1114, Nov. 1969.

平衡のとれた計算機システムにおいて，装置に適合した要求を出すための処理装置と記憶装置の相対的能力の関係を論じている．基本的な統計の理論に裏づけられた解が，平衡の原則に関する定理の形で示されている．デマンド・ページング・システムへの応用が示されている．

Denning, P. J.: "The Working Set Model for Program Behavior," *CACM*, vol. 11, no. 5, pp. 323-333, May 1968.

プログラム動作を取扱う“ワーキング・セットのモデル”を用いて，ページングによる記憶装置の動的な管理技法を実現することが目的である．ワーキング・セットとは“プログラムで最近使用されたページの集合”として定義されている．確率論的な議論によって，ページングのトラフィックなどの解析的な評価式が求められている．さらに，システムの要求に基づく資源割付けの方針を定式化するためにこのモデルが使用されている．

Denning, P. J.: "Third Generation Computer Systems," *ACM Computing Surveys*, vol. 3, no. 4, pp. 175-216. Dec. 1971.

本報告は，60年代の中期から現在に至る計算機システムの中で見られるオペレーティング・システムの基本原理を論評したものである．共通的なシステムの性質とプログラミングの方針が論じられている．また，記憶管理，プロセス管理，資源割付け，および保護の領域に基本的な問題点が示されている．註釈つきの文献表はこの領域の多数の最近の報告および本を含んでいる．

Denning, P. J.: "Thrashing: Its Causes and Prevention," *Proceedings, AFIPS, 1968, FJCC*, vol. 33, pp. 915-922.

スラッシング動作の議論において，記憶管理と処理装置の利用率との関係が明らかにされている．特に興味のある点は，処理装置の実効的な利用時間に対するページ読込み時間Tの影響である．最後に，Denning は3階層の記憶システムと小容量のページを用いることによってTを減少させることができると推論している（第2層記憶装置のアクセスによる遅延は最小であると仮定している）．

Denning, P. J.: "Virtual Memory," *ACM Computing Surveys*, vol. 2, no. 3, pp. 153-190, Sept. 1970.

仮想記憶の構成の方法に関する基本的な原理が論評されている．セグメンテーションとページングの対照は組込みの問題をよく理解させる．“機構”の議論からは方針一どこで，いつ，どのようにして情報にアクセスしたり格納を行なうのか一の議論が生まれる．次に，プログラム動作の検討の必要性を述べている．参照の局所性，ワーキング・セット，およびスラッシングの問題が発生し，関連が示される．この報告は，仮想記憶に対する現状を示し，将来の研究に対する手がかりを与える．広範な文献一覧表がつけられている．

Denning, P. J. and S. C. Schwartz: "Properties of the Working-Set Model," *CACM*, vol. 15, no. 3, pp. 191-198, March 1972.

ワーキング・セットの時間変化の性質に重点を置いて，数学的なモデルが示されている．ワーキング・セットの容量とページなしの確率との関係を予測するアルゴリズムが作られ，時間平均と統計的な議論とがうまく相補している．この解析には，平均のワーキング・セット容量，ページなしの確率，および相互干渉時間の分布の間に存在する関係が含まれている．

Dennis, J. B.: "Segmentation and the Design of Multiprogramming Computer Systems," *JACM*, vol. 12, no. 4, pp. 589-602, Oct. 1965.

セグメンテーションを行なっている番地空間の構成原理とその組込みとが論じられている．組込みについてはハードウェアとソフトウェアについて論じている．他の番地指定の方法についての比較も行なわれている．

Dennis, J. B. and E. C. Van Horn: "Programming Semantics for Multiprogrammed Computations," *CACM*, vol. 9, no. 3, pp. 143-155, March 1966.

セグメンテーション，共用，プログラムの並行実行，および多重プログラミングを行なうシステムの開発に関する初期の文献の1つである．システムの原理を発展させ，保護と共用を可能にする階層的なディレクトリを採用している．

Dijkstra E. W.: "Cooperating Sequential Processes," *Programming Languages*, Academic Press, 1968, pp. 43-111.

セマフォアは，プロセスの活動を整合して資源管理を可能にする理論的／物理的機構である．これの使用法については例を用いて詳細に説明されている．すくみを防止するための制限つきの割付け法についても論じられている．

Dijkstra, E. W.: "Solution of a Problem in Concurrent Programming Control," *CACM*, vol. 8, no. 9, pp. 569-570, Sept. 1965.

課せられた問題は，どんなときでもN台の計算機の内の1台だけしかクリティカルな部分の実行を認めないような制御機構を決定することである．計算機は繰返しの計算を行なっていて，共通の記憶域をとおして相互に通信することができる．この独占使用のクリティカルな部分は割込みを禁止して実行している期間に当る．解と証明とが示されている．この短い報告は，良心的で価値のある研究の古典的な一例である．

Dijkstra, E. W.: "The Structure of the T. H. E. Multiprogramming System," *CACM*, vol. 11, no. 5, pp. 341-346, May 1968.

分散型オペレーティング・システムの設計と開発の概説を個性的に行なっている．このシステムは基本的に，階層的ソフトウェア・システムの中で協同しているプロセスの一群を多重プログラミングするものである．各階層は，論理設計，効率的な動作，およびシステムの証明を容易にしようとする目的で特定の役割りを果たしている．プロセス同期の基本的な性質はセマフォア動作によって記述されている．本報告は，このような設計技法は大規模システムの開発に有効であると提案している．

Donovan, J. J.: *Systems Programming*, McGraw-Hill Computer Science Series, McGraw-Hill Book Co., New York, 1972. 訳書：池田克夫訳・**システム・プログラム**，日本コンピュータ協会，1974.

本書は，システム・プログラミングに必要な事項を包含している．計算機の構成，アセンブラ語，アセンブラ，マクロプロセッサ，ローダ，高水準言語，形式的体系，コンパイラ，およびオペレーティング・システムの章を含んでいる．例題は IBM 360-370 に基づいており，各章には演習問題がつけられている．

Donovan, J. J. and S. E. Madnick: *Pedagogical and Research Aids for Software Engineering*, to be published, 1974.

本書は，過去6年間にわたって MIT や他の大学，会社において計算機ソフトウェアの教育に使用された環境，特別の道具，および教育理念を論じている．特に，本書は次に示す項目に関して教師用テキスト，学生のレポート，およびマニュアルなどの資料を含んでいる：

1. 演習問題とその環境的な条件
   a．機械語レベルでの 360 のシミュレータ
   b．アセンブラ語の問題と採点プログラム
   c．入出力割込み処理の演習
   d．ジョブ・スケジューラ，ページング，および装置ハンドラに関する問題，およびこれらの問題に対する解答を判定するシミュレータ
   e．大規模な設計と組込みの問題——たとえば，オペレーティング・システムやコンパイラの主要部分を作成したり，これらのプログラムを実行するための環境
2. 学生のプログラムを実行する計算料金を低減させるための機構
   a．OS/360 で実行し大幅に経費を低減させるモニタ
   b．高速 PL/I(F) コンパイラ

Drake, A. W.: *Fundamentals of Applied Probability Theory*, McGraw-Hill Book Co., New York, 1967, Chap. 5.

確率論の基礎がわかりやすく述べられている．マルコフ・モデルについては優れた章である．

Earl, D. B. and F. L. Bugly: "Basic Time Sharing: A System of Computing Principles," *Second ACM Symposium on Operating Systems Principles*, Princeton University, pp. 75-79, 1969.

タイムシェアリングのオペレーティング・システムについて述べるための用語が示されている．言葉の定義には，計算機の設計原理を述べるために使用することができる基本的なシステム構成に関するものが含まれている．

Easton, W. B.: "Process Synchronization without Long-Term Interlocks," *Third ACM Symposium on Operating Systems Principles*, Stanford University, pp. 95-100, Oct. 1971.

クリティカル・セクションの必要性を排除する判定機構を提案している．共用されている対象物には番号が割付けられる．この番号は，実行中のプロセスで変更が行なわれたかどうかを判定するために使用される．共通の情報のためのロックとアンロックのためのスケジューリングはもはや不要となる．本報告には，例題と，この原理を用いてシステムを構成するための理念が含められている．

Erwin, J. D. and D. E. Jensen: "Interrupt Processing with Queued Content-Addressable Memories," *Proceedings, AFIPS, 1970, FJCC*, vol. 37, pp. 621-627.

オペレーティング・システムにおける要求処理時間を最小にする割込み処理機構が述べられている．割込み処理装置の動作を行なうために行列を管理して内容により番地指定を行なう記憶装置が用いられている．

Estrin, G., R. R. Muntz, and R. C. Uzgalis: "Modeling, Measurement, and Computer Power," *Proceedings, AFIPS, 1972, SJCC*, vol. 40, pp. 725-738.

この評論的な報告は，コンピュータ・ユーティリティの能力を評価するための規準を選択している．これは，システムのモデルや，知能的な測定法，および利用者のインターフェスにおいて得られた結果を議論するための背景となる．計算機システムの設計と組込みにおいては，モデルに大きく依存することを示している．

Fano, R. M.: "On the Social Role of Computer Communication," *IEEE*, vol. 60, no. 11, pp. 1249-1253, Nov. 1972.

本報告は興味ある概説で，一読の価値がある．

Feller, W.: *An Introduction to Probability Theory and Ist Applications*, vol. 1, 3rd ed., John Wiley & Sons, Inc., New York, 1968.

確率論の基礎に関する最も完全な参考書の１つである．本書は広いレベルで読むことができる．理論と例題によって基礎的な深い理解ができる．連続的な時間のマルコフ・モデルを詳細に議論している．

Feieritag, R. J. and E. I. Organick: "The Multics Input/Output System," *Third ACM Symposium on Operating Systems Principles*, Stanford University, pp. 17-23, Oct. 18-20, 1971.

一般的な入出力要求を適当な装置のための特定の命令に変換する装置インターフェース・モジュールについて述べている．MULTICS ではこのサブシステムのおかげで，システム・プログラマが IOCS の動作にほとんど関与しなくてもよい．MULTICS における応用例が示されている．

Fields, S.: "Silicon Disk Memories Beat Drums," *Electronics*, pp. 85-86, May 24, 1971.

この報告は，新型の高速（たとえば 100 マイクロ秒）直接アクセス記憶装置の例を示している．

Fine, G. H. and P. V. McIsaac: "Simulation of a Time-Sharing System," *Management Science*, vol. 12, no. 6, pp. B180-B194, Feb. 1966.

Fontao, R. O.: "A Concurrent Algorithm for Avoiding Deadlocks," *Third ACM Symposium on Operating Systems Principles*, Stanford University, pp. 72-79, Oct. 18-20, 1971.

すくみを防止するアルゴリズムを"同時実行"するには，割付けに先立ってページの要求を決定する効率のよい機構を用意すればよいことが示された．このアルゴリズムでは，各プロセスの種別によって要求される資源の最大数を知ることが必要で，動的な動作情報を使用する．実際のアルゴリズムの数学的表現（要求後の同時性）と正しさの証明が示されている．計算の手間は，Haberman のアルゴリズムと同様にNの２乗の位であることがわかっている．

Foster, C. C.: "An Unclever Time-Sharing System," *ACM Computing Surveys*, vol. 3. no. 1, pp. 23-48, March 1971.

Massachussetts 大学のシステムは，広い地域から無制限に計算機にアクセ

スさせるタイムシェアリング・システムの組込み方法という点ではあまり明解ではないかもしれないが，教育的で有益なものである．組織的な設計と CDC 3600への組込みの手法は設計方針の理由づけの背景とともに詳細に述べられている．本報告書の価値は，原理を論じるのに計算機と独立に行なっていることにある．

Fotheringham, J.: "Dynamic Storage Allocation in Atlas Computer Including an Automatic Use of a Backing Store," *CACM*, vol. 4, no. 10, pp. 435-436, Oct. 1961.

Atlas システムを簡単に述べた初期の報告書である．Atlas システムはデマンド・ページングの先駆者である．

Freiburghouse, R. A.: "The Multics PL/I Compiler," *Proceedings, AFIPS, 1969, FJCC*, vol, 35, pp. 187-199.

MULTICS のコンパイラの設計目標と基本動作の概説が行なわれている．コンパイラの5つの段階——構文翻訳，宣言処理，意味翻訳，最適化，およびコード生成——はかなり詳しく説明されている．これらの機能に含まれる主要なモジュールを説明するために例が用いられている．この文献の内容は高度で複雑な相互関係を述べているので，システムの原理を知っている読者におすすめしたい．

Fuchi, K., H. Tanaka, Y. Manago, and T. Yuba: "A Program Simulator by Partial Interpretation," *Second ACM Symposium on Operating Systems Principles*, Princeton, N. J., pp. 97-104, 1969.

スーパバイザのプログラムをデバッグすることは依然としてシステム開発の主要な仕事である．本報告で述べられている"部分解釈"の技法は，管理プログラムを利用者プログラムとして実行して，実行のトレースを行ないながら特権命令の模擬を行なうことを提案している．他の命令は普通に実行される．迅速でハードウェアに独立な検査が可能なシミュレーション法が述べられている．この手法は仮想計算機システムと同様である（CP-67, VM/370 など）．

Funchel, K. and S. Heller: "Consideration in the Design of a Multiple Computer System with Extended Core Storage," *First ACM Symposium on Operating Systems Principles*, Gatlinburg, Tenn., Oct. 1967.

番地指定はできるが命令実行はできない大容量高速乱アクセス記憶装置を応用してシステム性能を改善することが論ぜられている．この研究の結果は，

Brookhaven 国立研究所における拡張コア記憶装置を共用した2台の CDC 6600 システム構成に基づいている．

Gaines, R. S.: "An Operating System Based on the Concept of a Supervisory Computer," *Third ACM Symposium on Operating Systems Principles*, Stanford University, pp. 17-23, Oct. 18-20, 1971.

モニタの機能モジュールを設計するための面白い手法が示されている．説明と例とを用いて"ソフトウェアで構成したスーパバイザ計算機"の概念を発展させている．主要原理は，複雑な仕事を実行するのに基本機能を用いて他のプロセスと組合せることができるような小さくて安定なスーパバイザを持つことである．プロセス間通信の機能についても論じられており，重要な役割りを果たしている．

Goldberg, R. P.: "Architecture of Virtual Machines," *Proceedings, AFIPS, National Computer Conference*, vol. 42, pp. 309-318, 1973.

本報告書は，仮想計算機で実行しているプロセスによって資源の写像と番地指定を行なわせるようなモデルを展開している．このモデルの性質を推論することによって，現実の仮想計算機システムの考えを明瞭にしている．このモデルから得られた最も重要な結論は，適当な組込みによって仮想計算機のモデルの直接的で自然な組込みとして，ハードウェアの仮想化が実現できるということである．このハードウェアの仮想化の性質が論ぜられ，具体的な例を用いて，それの動作が説明されている．

Graham, R. M.: "Protection in an Information Processing Utility," *First ACM Symposium on Operating Systems Principles*, Gatlinburg, Tenn., Oct. 1967. (Also *CACM*, vol. 11, no. 5, p. 365, May 1968.)

システムの環境に関する興味ある議論とその結果必要と考えられる個人の秘密保護を動機として，保護機構の性質が概説されている．選ばれた保護の問題に対する解は，Dennis によって提案されたセグメンテーションを行なう計算機システムのモデルに基づいている．各セグメントはリングに基づくアクセス権を与えられるので，このモデルを拡張してリング番号を持たせることにより，層になった保護を行なうことができる．このハードウェア機構の議論に加えて必要となるソフトウェアについても概説されている．MULTICS におけるこの機構と実際の組込みの詳細な議論は Schroeder および Saltzer が報告している．

Graham, G. S. and P. J. Denning: "Protection—Principles and Prac-

tice," *Proceedings, AFIPS, 1972, SJCC*, vol. 40, pp. 417-430.

保護の必要性が示され，互に疑わしいサブシステムの間で協同作業を行なうことができるようにするシステムの抽象的なモデルを示して論じている．この研究は主として Lampson のモデルに基づいているが，幾つかの重要な点について研究を拡張している．保護に関する理論的な基礎を示し，組込みの問題点を論じている．解析は現在の知識の範囲で行なっている．より詳細について興味のある人のために有益な文献表がつけられている．

Gustavson, F. G.: "Program Behavior in a Paging Environment," *Proceedings, AFIPS, 1968, FJCC*, vol. 33, pt. 11, pp. 1019-1032.

IBM の M44/44X というページングを行なう実験的なタイムシェアリング・システムに関する研究の要約と，仮想記憶の原理の研究が行なわれている．この報告は使用したハードウェアを主題とはしているが，測定技法に関する情報，ページングを行なっている場合のプログラムの動作，記憶管理（自動的なものとプログラマの制御によるものの比較），およびデータの相互関係などは同様なシステムの研究を行なおうとする者にとっては価値がある．

Habermann, A. N.: "Prevention of System Deadlocks," *CACM*, vol. 12, no. 7, pp. 373-377, July 1969.

現在の資源の保有状態に基づいて要求に応じようとする多少共制限の少ない割付けアルゴリズム，およびすくみを生じる確率とを示している．システムをベクトルまたは行列によって表現することに基づいている．このアルゴリズムは，すくみを防止するための必要にして十分な条件であることが示されている．要求に応答する順番や安全でない要求に対する処置は，システムに依存して決定すべきものであると述べられている．

Habermann, A. N.: "Synchronization of Communicating Processes," *Third ACM Symposium on Operating Systems Principles*, Stanford University, pp. 80-85, Oct. 1971. (Also *CACM*, vol. 15, no. 3, pp. 171-176, March 1972.)

プロセス間通信システムの設計原理に"クリティカル・セクション"の考えが用いられている．この設計によって，微妙なデータの書込みや要求のための手続きを2つ以上のプロセスが同時に実行することを防止し，一方では，メッセージに対して同時読出しができるようになる．理論的な展開と証明が本報告の主体となっている．

Hansen, P. B.: "Short Term Scheduling in Multiprogramming Sys-

tems," *Third ACM Symposium on Operating Systems Principles*, Stanford University, pp. 101-105, Oct. 1971.

"スケジューリングの最も基本的な階段においては，すべてのシステム・プロセスを同時に取扱う"という設計原理が表面に出されている．この研究は Dijkstra, Lampson, および Saltzer の研究を発展させたものである．その手法は，同じ著者が以前に行なった，入出力と計算のプロセスを区別してスケジューリングする RC 4000 システムに関する研究よりも発展性のあるものである．短時間および中間時のスケジューリング(ハードウェア管理と利用者管理)に関する記述は，Algol 60 に類似した言語 Pascal によって行なわれている．クリティカル・セクションの考え方を基本としている．

Hansen, P. B.: "The Nucleus of a Multiprogramming System," *CACM*, vol. 13, no. 4, pp. 238-241, April 1970.

プロセスの実行という考えに基づく階層的なオペレーティング・システムの基本的な性質が示されている．この議論は RC 4000 システムに基づいており，システムの核，内部／外部プロセス，プロセス通信，およびプロセスの階層に関する議論が含められている．主な点は，この種の融通性のあるシステムは，多目的で取換えのきくオペレーティング・システムの構成を可能にするということである．

Hargraves, R. F. Jr. and A. G. Stephenson: "Design Considerations for an Educational Time-Sharing System, *Proceedings, AFIPS, 1969, SJCC*, vol. 34, pp. 657-664.

Dartmouth タイムシェアリング・システムについて論じている．

Hatfield, D. J.: "Experiments on Page Size, Program Access Patterns and Virtual Memory Performance," *IBM Journal of Research and Development*, vol. 16, no. 1, pp. 58-66, Jan. 1972.

ページ容量とアクセスの型に関する種々の実験結果が示されている．"ページ容量"の異状現象を示す経験的な証拠が示されている．この現象は．ページ容量を半分にしたとき，ページ入出力が極端に増加するものである（2倍以上になることもある)．この結果は，IBM システム／360 の実際のプログラムの命令の軌跡から求められている.（この現象のためにページ入出力に制限を加えた Madnick の研究を参照 (1973).）

Hatfield, D. J. and J. Gerald: "Program Restructuring for Virtual Memory," *IBM Systems Journal*, vol. 10, no. 3, pp. 168-192, 1971.

本報告は，サブルーチンの順番を変更することによって，デマンド・ページングを行なっているシステムの性能を改善する技法を示している．この順番づけは，頻繁に使用されるサブルーチンを一まとめにして同じページに置くようにするものである．

Havender, J. W.: "Avoiding Deadlocks in Multitasking Systems," *IBM Systems Journal*, vol. 7, no. 2, pp. 74-84, 1968.

システムのすくみを防止するために利用できる技法を説明するために，多重タスクを行なうシステムにおける資源割付けの問題を述べ，これらの技法の評価を行なっている．データ・セット，装置および記憶域を割付けるタスクの例としてジョブ・イニシェータが用いられている．ジョブの設定と実行においてすくみを防止するためには，ディスクの記憶域，主記憶域，および装置に対する要求の順番を定める番号づけの方式が有効であることが示されている．

Hellerman, H. and H. J. Smith, Jr.: "Throughput Analysis of Some Idealized Input, Output, and Compute Overlap Configurations," *ACM Computing Surveys*, vol. 2, no. 2, pp. 111-118, June 1970.

多重プログラミング動作に対するこの教育的手法によって，入力，出力，および計算時間が理想的な場合の性能評価式を求めてシステム動作の解析を行なっている．使用している性能評価の指数はシステムの仕事量である（システムの中でジョブを完了するために使用した時間の逆数）．研究されている2，3の場合を実際に追試してみることによって多重プログラミングの原理をよく理解することができる．

Ho-Nien, Liu, W. S. Peck, and P. T. Pollard: "Resource Management Subsystem for a Large Corporate Information System," *Proceedings, AFIPS, 1969, FJCC*, vol. 35, pp. 441-461.

汎用の計算機システムが各方面において使用された結果，各方面において計算力が必要であるという報告が多数行なわれている．本報告は，共同利用される情報システムの複雑なデータ要求を効率よく処理するためのサブシステムを，装置の有効利用を計る資源管理に組込むことを述べている．この考え方はこれまでに用いられていた通常のシステム設計法を包含しているので，バッチおよびオンラインの使用の両方に適用することができる．

Hoffman, L. J.: "Computer and Privacy: A Survey," *ACM Computing Surveys*, vol. 1, no. 2, pp. 83-103, June 1969.

この報告の動機は，計算機応用の範囲を拡大しようとすることである．"公

共のデータ・バンクの格子”——微妙な情報のアクセスの制御と倫理的な問題点が最初に論じられている．次に，情報の制御のために利用できる技法が論評されている．方式は長所と共に短所も述べられている．個人の秘密を守るという分野で成功した手法は，多数の手法の中でも，パスワード，擬似乱数によって符号化したメッセージ，および呼出し応答システムなどがある．将来の研究分野についても概説している．

Holt, R. C.: "Some Deadlock Properties of Computer Systems," *ACM Computing Surveys*, vol. 4, no. 3, pp. 179-196, Sept. 1972.

本報告の目的は，計算機システムにおけるすくみの性質の理論的研究の方法論を示すことである．目標は，より信頼度の高い計算機システムを構成することである．報告には，プロセス，すくみ，再使用可能な資源，および消費してしまう資源に関する概念が示されている．問題点を示すために有意義な例が用いられている．システムの形式的なモデル化が，グラフ理論を用いて紹介されている．一般的な資源システムの研究が行なわれており，すくみが起こるための必要十分条件が示されている．この領域の研究の現状を紹介するのに有用で優れた文献表がつけられている．

Horning, J. J. and B. Randell: "Process Structuring," *ACM Computing Surveys*, vol. 5, no. 1, pp. 5-30, March 1973.

“プロセス”という言葉は人によって異なった意味に用いられていた．本報告にはプロセスの分類と，妥適な定義が行なわれている．システム設計と文書化の複雑さを克服する手段として，プロセスの組合せと抽象化が行なわれている．階層的な構成は，より一般的な解析法の一部であることが示されている．本報告はプロセスとその環境を完全に記述している．

Husson, S. S.: *Microprogramming: Principles and Practice*, Prentice-Hall, Englewood Cliffs, N. J., 1970.

マイクロプログラミングの原理と利用法の総合的な紹介である． IBM 360/50, Honeywell 4200, および RCA Spectra 45 などの幾つかのマイクロプログラム計算機の詳細な事例研究が行なわれている．

Huxtable, D. H. R., and M. T. Warwick: "Dynamic Supervisors—Their Design and Construction," *First ACM Symposium on Operating Systems Principles*, Gatlinburg, Tenn., Oct., 1967.

計算機のスーパバイザに関する詳細な記述が行なわれている．この種のモジュールの組込み技法本報告の主要部分である．本報告は，システム・スーパバ

イザの設計マニュアルとして用いることができる．この報告の実用性は，実際のディスク・オペレーティング・システムのスーパバイザの設計と組込みを内容としている点であろう．

IBM, 1: *IBM System/360 MVT, Control Program Logic Summary,* Form Y28-6658-0.

IBM, 2: *IBM System/360: Principles of Operation,* Form GA22-6821.

IBM, 3: *IBM System/360: System Summary,* Form A22-6810.

IBM, 4: *IBM System/370: Principles of Operation,* Form GA22-7000.

IBM, 5: *IBM System/370: System Summary,* Form GA22-7001-1.

IBM, 6: *IBM System/360: Assembler Language programming,* Form GC28-6514.

IBM, 7: *IBM System/370: OS/VS and DOS/VS Assembler Language,* Form GC33-4010.

IBM, 8: *IBM System/360: Operating System, Planning for Multiprogramming with a Fixed Number of Tasks,* Version II (MFT II), Form C27-6939-0.

IBM, 9: *A Guide to IBM System /370, Model 155,* Form GC20-1729, pp. 11-18.

IBM. 10: *A Guide to IBM System/370, Model 165,* Form GC20-1730, pp. 14-24.

IBM, 11: *IBM System/370 Model 155: Functional Characteristics,* Form GA22-6942.

IBM, 12: *IBM System/370 Model 165: Functional Characteristics,* Form GA22-6935, pp. 14-16.

IBM, 13: *Introduction to OS/VS2 Release* 2, Form GC28-0661.

IBM, 14: *IBM Virtual Machine Facility/370 Introduction,* Form GC20-1800.

IBM, 15: *Planning Guide OS/VS2,* Form GC28-0600.

IBM, 16: *IBM System/360 Operating System: MVT Guide,* Form GC28-6720.

IBM, 17: *Introduction to Virtual Storage in System/370,* Form GR20-4260.

IBM, 18: *IBM System/360 and System/370 Bibliography,* Form GA22-6822.

IBM Cambridge Scientific Center: *CP/CMS Program Logic Manual*, IBM, Cambridge, Mass., 1969.

IBM 360/67 に組込まれた CP/CMS の説明である．これは仮想計算機の優れた組込みである．

Johnson, David S.: "Near-Optimal Bin Packing Algorithms," *MIT Project MAC Report TR-109*, Cambridge, Mass., June 1973.

本報告は，第1適合および最適適合を含めて，各種の動的分割割付けアルゴリズムを述べている．他のアルゴリズムについても研究している．平均的な動作の場合の経験則にもついて述べている．

Jones, P. D.: "Operating System Structures," *IFIP Congress Proceedings*, North-Holland Publishing Co., Amsterdam, 1969, pp. 525-530.

本報告は，代表的なオペレーティング・システムについて，ハードウェア，主要機能，および各機能の調整などの設計上の特徴を比較している．研究したのは CDC 6600/Chippewa, OS/360，および Atlas のスーパバイザである．

Josephs, W. H.: "An On-Line Machine Language Debugger for OS/360," *Proceedings, AFIPS, 196, FJCC*, vol. 35, pp. 179-186.

この実用的なシステムは DYDE と呼ばれ，バッチ・ジョブのジョブ制御言語を通じて呼出される．たとえば，デバッグ機能は IBM 2260 ディスプレイを通じて利用することができ，DYDE に制御が渡ったときに起動される．デバックはブレック・ポイントを設定することにより行なう．実際のデータがディスプレイにて表示される．デバッギング・システムについて興味ある組込みの詳細が示されている．

Kahn, G.: "An Approach to Systems Correctness," *Third ACM Symposium on Operating Systems Principles*, Stanford University, pp. 86-94, Oct. 1971.

小規模なオペレーティング・システムの記述と定理の証明に関するグラフ理論的な研究が行なわれている．グラフの技法によって，トークンの流れ図，通信線の合併，行列，節点，および入力／出力が説明されている．小規模システムに関しての証明の例によってこの方法論の有効性を示している．

Kapur, G. K.: *IBM 360 Assembler Language Programming*, John Wiley & Sons, Inc., New York, 1970.

初歩から，高度な原理とプログラム技法にわたってよく書けた本である．例

はディスク・オペレーティング・システムから引かれている.

Karush, A. D.: "Two Approaches for Measuring the Performance of Time Sharing Systems," *Second ACM Symposium on Operating Systems Principles*, Princeton University, pp. 150-166, 1969.

タイムシェアリング・システムに対する2つの技術評価の尺度（刺激／応答とソフトウェアのわな）が述べられ，ADEPT タイムシェアリング・システム（SDC 作成）に適用している.

Katz, J. H.: "Simulation of a Multiprocessor Computer System," *Proceedings, AFIPS, 1966, SJCC*, vol. 28, pp. 127-139.

IBM の直接結合システムを発表した報告である.

Katzan, H., Jr.: "Operating Systems Architecture," *Proceedings, AFIPS, 1970, SJCC*, vol. 36, pp. 109-118.

オペレーティング・システムの構成についてよくまとめてある.

Katzan, H., Jr.: "Storage Hierarchy Systems," *Proceedings, AFIPS, 1971, SJCC*, vol. 38, pp. 325-336.

各種の記憶管理とファイル・システムについてよくまとめてある.

Kay, I. M.: "An Over-the-Shoulder Look at Discrete Simulation Languages," *Proceedings, AFIPS, 1972, SJCC*, vol. 40, pp. 791-798.

この短い報告には，"シミュレーション・システムの民俗学に関する顧問の見解"が示されている．議論されている言語は型式と最初の開発者とによって分類されている．シミュレーションの実行に興味のある読者にとっては，これらの言語の広範な応用が示されている.

Kleinrock, L.: "A Continuum of Time-Sharing Scheduling Algorithms," *Proceedings, AFIPS, 1970, SJCC*, vol. 36, pp. 453-458.

2変数システムによって顧問の優先順位を規定するスケジューリング・アルゴリズムの1つのクラスが一般化され，実際的なモデルを用いて論じられている．選ばれた変数はサービス時間に基づくもので，これらの相対的な値は，FCFS から RR に至る広範なスケジューリング・アルゴリズムとなる．この解析技法によって，初期の2つの方法の間に位置する利己的な RR アルゴリズムの解析を行なうことができる.

Knuth, D.: *The Art of Computer Programming: Fundamental Algorithms,*

vol. 1, Addison-Wesley, Reading, Mass., 1968.

基本的なプログラム技法と情報構造が含まれている．本書はシステム・プログラムに興味ある人にとっては価値あるものである．リスト，樹，および多重リンクの構造について詳細に述べている．各種の動的な分割割付け記憶管理のアルゴリズムが示され，比較されている．

Kulsrud, H. E.: "Helper: An Interactive Extensible Debugging System," *Second ACM Symposium on Operating Systems Principles*, Princeton University, pp. 105–111, 1969.

デバッグ技術，特に会話型のグラフィック・システムを用いた技法の進歩が論じられている．Helper の構造が議論の主題となっている．コンパイラであり，シミュレータであり，デバッグ用のソフトウエアであり，制御用のパッケージでもある Helper の動作が実際の計算機の動作を通じて説明されている．

Lampson, B. W.: "A Scheduling Philosophy for Multiprocessing Systems," *CACM*, vol 11, no. 5, pp. 347–360, May 1968.

処理装置管理の基本的な問題が示されている．まず，処置の多重化の概念を理解するためにプロセスの性質が述べられている．次に，多重化を組込むために必要なシステムの機能に論じられている．これらには，優先権の方式，実行可リストの管理，時間のスケジューリング，およびロックが含まれる．論じられている主要な考え方の幾つかは最初 Saltzer の論文によって述べられたものである．

Lampson, B. W.: "Dynamic Protection Structures," *Proceedings, AFIPS, 1969, FJCC*, vol. 35, pp. 27–38.

コンピュータ・ユーティリティにおいて，保護を行なうモジュール自体はいかに保護されるのか．Lampson が研究したのはこの問題である．すなわち，利用者プログラムを実行するプロセスはいかにして区別され，生成され，破壊され，使用され，共用されるのであろうか．内容は，研究の方法論，能力，領域，およびアクセス・キーなどの概念によって構成されている．

Lampson, B. W.: "Protection," *Proceedings of the Fifth Princeton Conference on Information Sciences and Systems*, pp. 437–443, 1971.

保護とアクセス制御に関して現在のシステムの特性を紹介している．次に，一般的な保護方式を組込むためのモデルとして，"メッセージ・システム"が述べられている．領域という概念が，相互にアクセスし通信する情報の範囲を示

すものとして紹介されている．最後に，このモデルの能力を増すために，外部から領域を制御する“目的システム”が展開されている．組込み技法の討論も含められている．この短い論文は，動的な保護の実用的なモデルを作る基礎的な仕事である．

Lehman, Meir M. and Jack L. Rosenfeld, "Performance of a Simulated Multiprogramming System," *Proceedings*, AFIPS, *1968, FJCC*, vol. 32, pp. 1431-1442.

この報告書は A. L. Scherr によって行なわれた実験結果（"Analysis of Main Storage Fragmentation," *Proceedings of IBM Symposium on Storage Hierarchy Systems*, TR. 00. 1556, pp. 159-174, Dec. 30, 1966）を述べている．Scherr は，多数の 360/65 の計算機システムにおいては65%もの待合せ率が普通であることを示している．Lehman と Rosenfeld は，システムの入出力待合せ時間に対する多重プログラミングの効果について実験を重ねている．

Lichtenberger, W, W. and M. W. Pirtle: "A Facility for Experimentation in Man-Machine Interaction," *Proceedings, AFIPS, 1965, FJCC*, vol. 27, pp. 185-196.

XDS-940 タイムシェアリング・システムについて述べている．

Linde, R. R., Weissman, and C. Fox: "The ADEPT-50 Time Sharing System," *Proceedings, AFIPS, 1969, FJCC*, vol. 35, pp. 39-50.

タイムシェアリング・システムのモニタの構成法と組込みについて詳細に論じている．このシステムは，政府のために SDC によって IBM360 に組込まれた試作システムであり，汎用のタイムシェアリング，データ管理システムである．この設計はモジュール化を行なって，分散型のモニタとなっている．モニタ設計の優れた概説が行なわれている．

Liptay, J. S.: "Structural Aspects of the System/360 Model 85: II. The Cache," *IBM Systems Journal*, vol. 7, no. 1, pp. 15-21, 1968.

IBM は小容量高速度のバッファを用いて，Wilkes の唱えたスレーブ・メモリの原理を製品の組込み，主記憶の遅いサイクル時間を短縮することに成功した．この結果 CPU の利用率が改善された．モデル85の構成の基になった研究が詳細に述べられていて，Wilkes が述べた論点を確認している．

Liskov, B. H.: "A Design Methodology for Reliable Software Systems," *Proceedings, AFIPS, 1972, FJCC*, vol. 41, pt. 1, pp. 191-200.

本論文は2つの仕事から構成されている．まず，システムの設計が満たすべき規準について論じている．この議論では責任のレベル——階層的な機能と構造化したプログラミング——が明らかにされている．次に，設計者が目標を達成してシステムを組込むための道しるべが示されている．この道しるべは，効果的なシステムのモジュール化を達成することを目ざしている．本報告のような論文は，トップ・ダウンの設計および構造化プログラミング法の“改良，拡張，および明確化”の第一歩である．

Liskov, B. H.: “The Design of the Venus Operating System,” *CACM*, vol. 15, no 3, pp. 144-156, March 1972.

マイクロプログラム—ソフトウェア方式のオペレーティング・システムの融通性の価値が，設計原理を理解させようとする説明によって示されている．本研究の主たる目的はシステム・ソフトウェアの開発に対するシステム構成の効果（可変のファームウェアに基づく）を評価しようとするものであった（システムを開発して機能を簡単に定義することが可能である）．実際のシステムは，階層化したプロセス向きの環境，セグメンテーションによる仮想記憶，および16個のプロセスの多重プログラミングなどの高度の資源管理を行なう第4世代のシステムの特性を備えている．本報告は，実用研究の現状を示す例としておすすめする．

Lucas, H. J., Jr.: “Performance Evaluation and Monitoring,” *ACM Computing Surveys*, vol. 3, no. 3, pp. 79-92, Sept. 1971.

計算機の性能評価およびモニタリングの研究は大変重要な問題であることからよく参照される．新しい方法論の発展している分野において，Lucasは目標と適用される技法との分類を試みている．たとえば，システムの評価の一般的な目的は次の3つであると述べている．

選択法の評価
性能の設計
性能の監視

次に，この目標を達する適合性に関して，各種の技法をまとめて表にしている．この技法は簡単に説明もされている．広範な知識を統一するために用いた手法は当を得ている．

MacDougal, M. H.: “Computer System Simulation : An Introduction,” *ACM Computing Surveys*, vol. 2, no. 3, pp. 191-209, Sept. 1970.

高水準言語を用いて（シミュレーション用の言語ではなく）シミュレーション・モデルを構成するための問題点が，ディスク・ベースの多重プログラミング・システムの研究を行なうことによって明らかにされている．可変長の事象リストの管理，共有資源のための行列，および統計データの集収がすべて説明されている．システム・シミュレーションの原理に興味を持つ人にとっては，よき出発点となろう．さらに，より進んだ研究のために，完全に註釈のついた文献表がつけられている．

Madnick, S. E.: "Multi-Processor Software Lockout," *Proceedings of the ACM National Conference* 1968.

本報告は，共用のデータベースの変更が一時に1台の処理装置に限って実行されるような多重処理装置システムにおける処理装置のアイドル時間について研究したものである．連続の時間をとるマルコフ・モデルを用いて，システム内の処理装置の台数の関数として処理装置ロックアウトの確率を求めている．

Madnick, S. E.: "Storage Hierarchy Systems," *MIT Project MAC Report TR-105*, Cambridge, Mass., 1973.

本報告では，多層の記憶階層の議論と解析が行なわれている．主な結果は，ページ容量の異状現象の定式化と，ページ容量を小さくしたときの悪影響を制限するためのアルゴリズム――項結合法――の開発である．

Madnick, S. E.: "Time-Sharing Systems: Virtual Machine Concept vs. Conventional Approach," *Modern Data*, vol. 2, no. 3, pp. 34-36, March 1966.

タイムシェアリングに対する仮想計算機の利用の特性と利点とを述べた初期の報告である．

Madnick, S. E. and J. W. Alsop: "A Modular Approach to File System Design," *Proceedings, AFIPS, 1969, SJCC*, vol. 34, pp. 1-14.

"階層的なモジュール" に基づくファイル・システムの設計法が示されている．結論としてファイル・システムの一般的なモデルが得られ，著者によって研究されたファイル・システムは，幾分抽象化が必要ではあるが，このモデルに適合することが認められている．

Madnick, S. E., and J. J. Donovan: "Application and Analysis of the Virtual Machine Approach to Information System Security and Isolation," *Proceedings of the ACM Workshop on Virtual Computer Systems*,

March 1973.

通常のオペレーティング・システムよりも安全性と隔離性をよくするためにオペレーティング・システムと仮想計算機モニタのインターフェースの分離が示されている.

Martin, W. A.: "Sorting," *ACM Computing Surveys*, vol. 3, no. 4, pp. 147-174, Dec. 1971.

本報告は，分類法を論評して，適用分野に対して現在求められている最適のアルゴリズムを示している．原理は100冊以上もの書籍や文献から選択されている．この領域の技法について学びたい人によき第1歩を与えるであろう.

Mattson, R. et al.: "Evaluation Techniques for Storage Hierarchies," *IBM Systems Journal*, vol. 9, no. 2, pp. 78-117, 1970.

デマンド・ページングのページ置換えアルゴリズムの性能を解析するためには，スタック処理が役に立つことが明らかにされている．FIFO, LRU, LFU, OPT, および乱置換えアルゴリズムの基本的な性質を明らかにしようという目的で理論的な観点から説明されている.

McFarland, C.: "A Language Oriented Computer Design," *Proceedings, AFIPS, 1970, FJCC*, vol. 37, pp. 629-640.

システムの中において，ハードウェア，モニタ，および計算機用の言語の相互の関係の適合性を最大にしようとする設計原理が提案されている．このような実験的研究が読者のために理論づけされている．設計目標の1つは，ハードウェア化したソフトウエア機能である．システム計画においてしばしば見過される相互関係の理解ができるようになる.

McKinney, J. M.: "A Survey of Analytical Time-Sharing Models," *ACM Computing Surveys*, vol. 1, no. 2, pp. 105-116, 1969.

計算機の時分割利用の原理が，この種のシステムの解析的なモデルの研究とそれに関する結論を論じるための基礎として，紹介されている．RR, FB, および外部的な優先順位などの，タイムシェアリングの応用に関連するサービス方針も基礎的事項の一部に含められている．研究と結論の考察は，タイムシェアリングのモデル化に興味を持つ解析者が共通に参照することができる．註釈つきの文献一覧表がつけられている.

Mclure, R. M.: "An Appraisal of Compiler Technology," *Proceedings, AFIPS, 1972, SJCC*, vol. 40, pp. 1-10.

設計を統一しようと試みている工場生産のコンパイラの概説が行なわれている．本報告は，構文解析，内部表現，表と表の管理，組込みの問題点の技法について簡単に触れている．コード生成と最適化についても簡単に述べられている．初めての読者に対しては，特定の問題に対して新しい技法や適切な技法のヒントが示されている．D. Gries 著の *Compiler Construction for Digital Computers*（訳書：牛島和夫訳・**コンパイラ作成の技法**，日本コンピュータ協会，1976）が，“さらに進んで研究するために優れた文献一覧表を含んだ，現時点での最良の書物である”として述べられている．

Meade, R. M.: "On Memory System Design," *Proceedings, AFIPS, 1970, FJCC*, vol. 37, pp. 33-44.

階層的な記憶システムの設計に対する理論的および統計的基礎が，経費一性能比という観点から説明されている．ブロック容量，バッファ容量，時間の制御アルゴリズム，転送比，および性能の尺度が論じられ，記憶システムの設計における主要な問題点と相互関係の丁寧な概説となっている．

Mealy, C. H. et al.: "Program Transferability Study," Rome Air Development Center, Griffiss AFB, Technical Report No. RADC-TR-68-341 (U. S. Clearinghouse for Federal Scientific and Technical Information より入手可能), Nov. 1968.

本報告は，プログラムの互換性について詳しく研究している．広範な言語プログラム，および環境が考察され，次の3つのレベルが提案されている：

1. 文書，プログラムの管理上の制御
2. 互換性を増すための現在のオペレーティング・システム／言語システムの拡張
3. “高度に互換性のある環境”の長期にわたる発展

それぞれの方法の利点と欠点が論じられ，勧告が行なわれている．

Molho, L.: "Hardware Aspects of Secure Computing," *Proceedings, AFIPS, 1970, SJCC*, vol. 36, pp. 135-143

論理的な誤動作や破壊，およびこれらの反作用の問題点が，IBM 360/50 システムのハードウェア特性を例に引いて示されている．この議論を基にして，誤りに対して安全な動作，誤り検出，およびデータ検査といった安全に計算を行なうためのハードウェアの設計が論じられている．システムの保全というむずかしい仕事に理解を与える報告である．

Morris, R.: Scatter Storage Techniques," *CACM*, vol. 12, no. 1, pp. 38-

44, May 1968.

ハッシュ符号について詳しく論じている．本報告は，方法論の総覧と組込みの詳細——ぶつかった欄の解決法——を示す優れた文献である．

Morrison, J. E.: "User Program performance in Virtual Storage Systems," *IBM Systems Journal*, vol. 12, no. 3, pp. 216-237, 1973.

本報告は，デマンド・ページングの性能に影響する利用者プログラムの要因を示している．性能を改善するためのプログラム技法が示されている．

Motobayashi, S., T. Masuda, and N. Takahashi: "The HITAC 5020 Time Sharing System," *Proceedings of the 24th National Conference of the ACM*, ACM Publication P-69, pp. 419-429, 1969.

MULTICSシステムで用いられた多くの概念を採用したり拡張したHITAC 5020 システムについて述べている．

Murphey, J. E.: "Resource Allocation with Interlock Detection in a Multi-task System," *Proceedings, AFIPS, 1968, FJCC*, vol. 33, pt. II, pp. 1169-1176.

すくみを起こさないような資源要求を検出する行列管理法が述べられている．組込みは，待合せマトリックスあるいは優先権マトリックスを用いて行なわれる．前者は単純なYES/NO の検出のみが可能で，すくみが起こるときには資源をすべて解放し，繰返し要求することが必要である．後者は，すくみが起こるときに解放する資源の最小の組を決定する．本報告は，例を用いて技法を示し，将来研究すべき領域をも述べている．

Needham, R. M.: "Protection Systems and Rrotection Implementations," *Proceedings, AFIPS, 1972, FJCC*, vol. 41, pt. I, pp. 571-578.

間接番地表の処理に基づいて計算機の主記憶中の情報を保護する方式が提案されている．この機構の採用はロックとキー方式の欠点を解析して決定された．表を用いる保護方式の要点が概説され，不正な引数の問題が論じられている．

Needham, R. M. and D. F. Hartley: "Theory and Practice in Operating System Design," *Second ACM Symposium on Operating Systems Principles*, Rrinceton University, pp. 8-12, Oct. 1969.

理論と通常の設計の橋渡しをねらって，スプーリングの設計，すくみ，および多重プログラミングの効率が述べられている．設計の研究に対してもっと成果が期待される領域が述べられている．

Neumann, P. G.: "The Role of Motherhood in the Pop Art of System Programming," *Second ACM Symposium on Operating Systems Principles*, Princeton University, pp. 13-18, 1969.

ソフトウエア・システム測定のための物指しが，単なる口先きの言葉でなく，実際に使用されることを期待して，提案されている．デバッグ，文書化，融通性，モジュール性，互換性などの望ましいシステムの機能が定義され，避けるべきものが示されている．表現はやさしく読みやすい．

Nielsen, N. R.: "Computer Simulation of Computer System Performance," *Proceedings, ACM National Conference, 1967*, pp. 581-590.

初期の IBM TSS/360 システムのシミュレーションを行ない，平衡がうまく取れていないシステムにおけるアイドル時間を明らかにしている．

Oppenheimer, G. an N. Weizer: "Resource Management for a Medium Scale Time Sharing Operating System," *First ACM Symposium on Operating Systems Principles*, Gatlinburg, Tenn., Oct. 1967.

平衡のとれたシステムの性能と資源の利用率を達成するための基本的な構成を求めようとして，RCASpectra 70/46 計算機のタスク・スケジューリングの基本特性が述べられている．GPSS 型の言語を用いたシミュレーションに基づいて，各種のスケジューリングやページング・アルゴリズムの評価が行なわれている．

Organick, E. I.: *The Multics Systems: An Examination of Its Structure*, The M. I. T. Press, Cambridge, Mass., 1972.

本書は，MULTICS に関する最初の総合的な書物である．MULTICS の完全なる参考文献も含められている．

Pankhurst, R. J.: "Program Overlay Techniques," *CACM*, vol. 11, no. 2, pp. 119-125, Feb. 1968.

自動的，半自動的，および手動のオーバレイ機構が，主要な仮想記憶管理技法として述べられている．例が半自動的なオーバレイ・プロセッサと CDC 6600 から引かれている．

Parmelee, R. P., T. I. Peterson, C. C. Sullivan, and D. S. Hatfield: "Virtual Storage and Virtual Machine Concepts," *IBM Systems Journal*, vol. 11, no. 2, pp. 99-130, 1972.

本報告は，CP-671 CMS の優れた記述と，仮想計算機のすべての特性に関

する註釈つき文献表を示している．

Presser, L. and J. R. White:　"Linkers and Loaders," *ACM Computing Surveys*, vol. 4, no. 3, pp. 149-169, Sept. 1972.

言語翻訳における結合とローディングの段階に関する本解説は，結合とローディングにおける"人間／計算機の通信の過程"の中心的な原理を紹介している．IBM 360 で行なわれているようなリンケージ・エディタと簡単な再配置型ローディングのモジュールとしての見方が中心となっている．独立に翻訳されたモジュールを結合する仕事が，例を用いて詳細に述べられている．オーバレイ処理，プログラムの変更，およびライブラリ参照などの2次的な機能についても述べられている．

Price, C. E.:　"Table Look Up Techniques," *ACM Computing Surveys*, vol. 3, no. 2, June 1971.

順次的，合併，2分，予測，および直接探索などの静的な表の探索に関する基本的な方法論が示されている．ハッシュ法の番地指定についても述べてある．本報告は，Martin が書いた表の管理の部分に収められ，Donovan の書物の第4章となっている．

Randell, B. and C. J. Kuehner:　"Demand Paging in Perspective," *Proceedings, AFIPS, 1968, FJCC*, vol. 33, pt. II, pp. 1011-1018.

デマンド・ページングの定義がなされ，次に，デマンド・ページング・システムの性能を改善するための技法を総覧している．ハードウェア的な手法は強力であるとしてしりぞけられ，ソフトウェア的な手法が詳しく論じられている．これらの技法には，プログラム負荷を調整したり，オペレーティング，システムの処理方法——スケジューリングと資源割付け——を調整することも含まれている．要約と結論は，ページング・システムの動作を理解するのに役立つ．

Randell, B. and C. J. Kuehner:　"Dynamic Storage Allocation Systems," *First ACM Symposium on Operating Systems Principles*. Gatlinburg, Tenn., Oct. 1967.

動的な記憶割付けシステムが，機能と組込み技法によって分類されている．基本的な4つの性質を用いて B5500, MULTICS, 360/67 などのシステムの自動的な記憶管理システムを研究している．記憶管理に使用されている各種のハードウェア機能の研究のための枠組を発展させている．

Reiter, Allen, "A Resource Allocation Scheme for Multi-User On-Line Operation of a Small Computer," *Proceedings, AFIPS, 1967, SJCC*, vol. 30, pp. 1-8, 1967.

本報告は，小規模な計算機（たとえば IBM 360/30）のために設計されたオペレーティング・システムについて述べている．このシステムは，一種の動的なセグメント・スワッピングを行なうソフトウェア技法に基づく面白い記憶管理方式を採用している．

Risko, F. D.: "New Horizons for Magnetic Bulk Storage Devices" *Proceedings, AFIPS, 1968, FJCC*, vol. 33, pt. II, pp. 1361-1368.

記憶装置について興味ある概説が示されている．これには大容量記憶の評価規準も述べられている．ハードウェア技術の現状が明らかにされ，レーザ記憶に対する動向が論じられている．本報告は，今日の状況との興味ある比較を行なっている．

Rodriguez-Rosell, Juan: "Experimental Data on How Program Behavior Affects the Choice of Scheduler Parameters,"*Third ACM Symposium on Operating Systems Principles*, Stanford University, pp. 156-163, Oct. 1971.

プログラムのワーキング・セットの特性（ワーキング・セット容量の経験的な測定）および処理装置スケジューリング変数（割当て時間）の選定との関係の報告である．システム・プログラムを完全に解釈するシミュレータを用いた実験によって各種の結果が求められ，図に示されている．

Rosen, S.: "Electronic Computers: A Historical Survey," *ACM Computing Surveys*, vol. 1, no. 1, pp. 7-36, March 1969.

計算機システムとその製造会社の興味ある，詳細な，要領のよい歴史は "読者が知りたいと思っていたこと" を端的に表わしているであろう．主たる内容は米国における発展に限っている．計算機のよく知られた3つの世代が明らかにされ，研究を広範囲に含む参考文献が示されている．

Rosin, R. F.: "Contemporary Concepts of Microprogramming and Emulation," *ACM Computing Surveys*, vol. 1, no. 4. pp. 197-212, 1969.

本報告は，仮説の計算機の制御機能を，マイクロプログラミングの技法を適用したり対比したりしながら解説的に述べている．マイクロプログラミングを行なっている幾つかのシステムについて述べている．さらに，将来の研究の領域が概説されている．

Rosin, R. F.: "Supervisory and Monitor Systems," *ACM Computing Surveys*, vol. 1, no. 1, pp. 37-54, March 1969.

コンピュータ・ユーティリティのオペレーティング・システムの機能の発展を年代順に詳細に述べている．記述の完全さは犠牲になっているが，この分野の研究を始めようとする人が読むのに適している．示された事項は，計算機の熟達者には必修事項である．

Ross, D. T., "The AED Free Storage Package," *CACM*, vol. 10, no. 8, pp. 481-492, Aug. 1967.

本報告は，AED-1 コンパイラ・システムの空き記憶域パッケージに設けられた動的記憶割付け機能を述べている．このパッケージによって，利用者は空き領域を幾つかのゾーンに分け，ゾーンごとに別個の割付けアルゴリズムを用いることができる．

Saltzer, J. H.: "Traffic Control in a Multiplexed Computer System," *MIT Project MAC Report TR-30*, Cambridge, Mass., 1966.

信頼性，アクセス可能性，および共用などの問題を知っているものとして，本論文は，オペレーティング・システムとコンピュータ・ユーティリティの中での機能を管理するための主要モジュール――トラフィック制御――の性質を論じている．この手法は解説的なものであり，今日の計算機システムの主要部分となっているプロセス間通信および多重処理装置の多重化に関する進歩した概念を示している．優れた管理方式が提案されており，この原理が MULTICS システムの主幹となっている．

Saltzer, J. H. and J. W. Gintell: "The Instrumentation of MULTICS," *Second ACM Symposium on Operating Systems Principles*, Princeton University, pp. 167-174, 1969.

本報告は，コンピュータ・ユーティリティの保守と研究のために使用される測定用の道具の総覧である．この中には，特殊ハードウェアによる時計とデータ・チャネル，汎用のソフトウェアによる探触子と記録の道具，および専用の測定機能が含まれている．デマンド・ページングの環境で用いられる特殊なハードウェア／ソフトウェアの測定装置について述べられている．

Sayer, D.: "Is Automatic 'Folding' of Programs Efficient Enough to Displace Manual?" *CACM*, vol. 12, no. 12 pp, 656-660. Dec, 1969.

本報告では仮想記憶システムのためのプログラム組換えが論じられている．自動的に組換えられた（折りたたまれた）プログラムの性能が計算機システム

で使いものになることが示され，この技法の利点が示されている．動的な機構が10年前のコンパイラと比較され，これに関連した方法が将来採用されることになろうと予想している．

Scherf, J.: "Data Security: A Comprehensive and Annotated Bibliography," master's thesis, M, I. T. Sloan School of Management, Cambridge, Mass., 1973.

Scherr, A. L.: "An Analysis of Time-Shared Computer Systems," *MIT Project MAC Report TR-18,* Cambridge, Mass., 1965.

CTSS のアルゴリズムと性能の詳しい解析が行なわれている．

Schroeder, M. D.: "Performance of the GE-645 Associative Memory While MULTICS Is in Operation," *ACM Workshop on System Performance Evaluation,* pp. 227-245, April. 1971.

MULTICS のセグメンテーションとページングについて論じてから，記憶装置の参照の番地変換を行なうために処理装置で用いられている連想記憶の評価が行なわれている．

Schroeder, M. D. and J. H. Saltzer: "A Hardware Architecture for Implementing Protection Rings," *CACM,* vol. 15, no. 3, March 1972.

リングという名前のハードウェア機構が，タイムシェアリングで多重プログラミング，多重処理装置，セグメンテーションを行なう仮想記憶計算機システム MULTICS における アクセス制御に都合のよい技法であることが提案されている (Graham)．議論は，Honeywell 645 計算機の改訂版の機能に基づいて行なわれている．しかし，この議論はセグメンテーションを行なうシステムの保護に興味を持つ人にとって価値のあるものである．本報告には，呼出しと復帰によって領域を変更する動作の原理も述べられている．ハードウェアの機構が示されている．

Schwartz, D. S.: "Multiprogramming in a Page on Demand Computer System: Performance Models and Evaluation," master's thesis, Department of Electrical Engineering, M. I. T., Cambridge, Mass., 1973.

本報告は，計算機の解析的なモデルおよびシミュレーションのモデルの研究の文献調査である．3つのモデル化技法によって，多重プログラミングの動作時間と平均の CPU 利用率の解析が行なわれている．結果はわかりやすく示され，モデル化技法が比較対照されている．

Schwartz, J., E. G. Coffman, and L. Weissman: "A General Purpose

Time-Sharing System," *Proceedings, AFIPS, 1964, SJCC,* vol. 25, pp. 397-411.

SDC のタイムシェアリング・システムについて論じている.

Sekino, A.: "Performance Evaluation of Multiprogrammed Time-Shared Computer Systems," *M. I. T. Project MAC Report TR-103,* Cambridge, Mass., 1972.

本報告は，実在の多目的計算機システムの完全なる解析を行なっている. MULTICS に対するモデルを確認しようとする意図を持って，タイムシェアリングの多重プログラム，多重処理装置，デマンド・ページング，仮想記憶計算機の一般的なモデルを展開している.

Senko, M. E., E. B. Altman, M. M. Astrahan, and P. L. Fehder: "Data Structures and Accessing in Data-base Systems," *IBM Systems Journal,* vol, 1. pp. 30-93, 1973.

データベース情報システムの解析が3部にわたって示されている. 第1部は，データベース・システムの発達を論じ，将来の発展と応用について述べている. 第2部は，情報の構造について論じ，新しい手法について紹介している. 第3部は，情報の記述と内部表現の枠組——データに独立なアクセスのモデル（DIAM）を示している.

Sevick, K. C., et al.: "Project SUE asa Learning Experience," *Proceedings, AFIPS, 1972, FJCC* vol. 41, pt. I, pp. 331-338.

この研究計画では，階層的なオペレーティング・システムの開発を統合して進展させる多くの設計目標を持っている. 状況報告として，このシステムの本質と構成法が述べられている. 文献表には，この計画で行なわれた他の技術報告についても述べられている.

Skatrud, R. O.: "A Consideration of the Application of Cryptographic Techniques to Data Processing," *Proceedings, AFIPS, 1969, FJCC,* vol. 35, pp, 111-119.

本報告は，データ処理システムで応用できるディジタル的な暗号技法を2つ述べている. ローマ時代以降の暗号法の背景が論じられている. 2つの方法はディジタル的な代入法と径路の置換えであり，これらがわかりやすく説明されている.

Spier, M. J. and E. I. Organick: "The MULTICS Interprocess Com-

munication Facility," *Second ACM Symposium on Operating Systems Principles,* Princeton University, pp. 33-91, Oct. 1969.

プロセス間通信機能が MULTICS の一部として動作している状況について説明している．プロセス間通信機能の基本が計算機に独立な型で述べられている．次に MULTICS における応用と，システム設計者のための応用が概説されている．

Spirn, J. R. and P. J. Denning: "Experiments with Program Locality," *Proceedings, AFIPS, 1972, FJCC,* vol. 41, pt. I. pp. 611-621.

プログラムの局所性の概念が定義され，この分野でこれまで行なわれた研究が示されている．参照の局所性の定義は内部的な局所性と外部的な局所性に分類され，内部的な局所性を測定するために，ワーキング・セットを用いて外部から測定能力を評価する研究が行なわれている．内部的な局所性のモデルについての論評がなされ，期待性能と実際に比較するための実験規準が求められている．比較のデータを生成するために例題プログラムが使用され，プログラムの局所性の評価としてはワーキング・セットのモデルが優れていることが述べられている．

Teorey, T. J.: "Properties of Disk Scheduling Policies in Multiprogrammed Computer Systems," *Proceedings, AFIPS, 1972, FJCC,* vol. 41, pt. I, pp. 1-12.

ディスクのスケジューリングの解析の分野でなされた研究について述べ，適当な文献をあげている．次にこれらの一般に研究されたスケジューリング・アルゴリズムの性能の解析（かっては，解析にシミュレーションを行なわねばならないことが多かった）結果の一部として，数学的な取扱いについて研究している．解析結果が，スケジューリングを行なうべきか否か，行なうとしたらこの応用にはどんな方針が最良かを決定するための情報となれば幸いである．述べられている方針は，オペレーティング・システムとハードウェアに関して論じられている．

Teorey, T. J. and T. B. Pinkerton: A Comparative Analysis of Disk Scheduling Policies," *Third ACM Symposium Operating Systems Principles* Stanford University, pp. 114-121, Oct. 1971.

シーク時間と待合せ時間の期待値に対する性能評価規準に基づいて，可動ヘッド・ディスクに対して，FCFS, SSTF, 走査，N 段階走査などの方針が比較されている．与えられた状況に対する最適の方針が選ばれている．本報告はこ

の現領域での現在の解析法を用いてできる完全な議論である．

Van Horn, E. C.: "Design for Asynchronously Reproducible Multiprocessing," *M.I.T. Project MAC Report TR-34,* Cambridge, Mass., 1966.

ロックを組込むためのソフトウェアの機構が示されている．

Van Horn, E. C.: "Three Criteria for Designing Computing Systems to Facilitate Debugging," *First ACM Symposium on Operating Snstems Principles,* Gatlinburg, Tenn., 1967.

プログラムの検査に関する利用者のインターフェースに重要と考えられる3つの規準が概説されている．これらは入力の記録と，入力の規定と，非同期的な出力である．この規準が概説され，デバッギングを助ける計算機設計の問題が論じられている．

Vareha, A. L., R. M. Rutledge, and M. M. Gold: "Strategies for Structuring Two Level Memories in a Paging Environment," *Second ACM Symposium on Operating Systems Principles,* Princeton University, pp. 54-59, 1969.

2層で直接実行可能な記憶装置の効果的な利用の原理が論じられ，そのアルゴリズムが考察されている．また，その使用経験が示されている．本報告は，360/67　タイムシェアリング・システムに基づいている．低速の第2層の大容量記憶装置を導入するとページングの回数を減少させ，性能が改善されることが示されている．

Varney, R. C.: "Process Selection in a Hierarchical Operating System," *Third ACM Symposium on Operating Systems Principles,* Stanford University, pp. 106-108, Oct. 1971.

プロセス・スケジューリングにおいて処理装置の共用に対する新しいモデルが示されている．多重プログラミング，実時間，およびタイムシェアリングなどの動作の柔軟なモデル化が可能である．アダプティヴなスケジューリングに利用するために，処理装置選択の樹の説明が行なわれている．

Watson, R. W.: *Timesharing System Design Concepts,* McGraw-Hill Computer Science Series, McGraw-Hill Book Co., New York, 1970. 訳書：相磯秀夫訳，**タイムシェアリング・システム**，日本コンピュータ協会 1974.

タイムシェアリング・システムの設計原理を示した優れた書物である．

MULTICS と SDS 940 の2つのシステムを論じている.

Weissman, C.: "Security Controls in the ADEPT-50 Time-Sharing System," *Proceedings, AFIPS, 1969, FJCC*, vol. 35, pp, 119-134.

政府および軍の極秘の情報を処理する Adept-50 タイムシェアリング・システム（Linde 参照）の安全性の制御について述べている. 安全に対する4つの目標が示されている. これらは次のものである：

利用者
端末装置
ジョブ
ファイル

各目標には安全性を示した記述がなされ, 資源の間のタイムシェアリング・システムでの安全制御を規定するアクセス権の集合論的なモデルが採用されている. 本報告はきわめて詳細に述べており, 読者に安心感を与えるものである.

Wilkes, M. V.: "Slave Memories and Dynamic Storage Allocation," *IEEE TC* vol. EC-14, no. 2, pp. 270-271, April 1965.

この短い報告は, 典型的なコア記憶装置の階層システムの実効速度を増すための技法を論じている. 主記憶よりはるかに高速の小容量および大容量のスレーブ・メモリを2次的な記憶システムとして用いる考え方を導入している. 応用分野が述べられ, ハードウェアを簡単にするための技法が概説されている.

Wilkes, M. V.: "The Growth of Interest in Microprogramming: A Literature Survey," *Computing Surveys*, vol. 1, no. 3, pp. 139-145, Sept. 1969.

マイクロプログラミングという言葉の起源が, 他の応用を試みようという人のために, (この技法の生みの親によって説明され, 定義されている. 次に, 関係の論文を簡単に論評して, この設計の手法が国際的に認識されていることを明らかにしている (55の文献を示している). これによって, 計算機の分野でのファームウェアの能力と融通性と将来性を認識することができる,

Wilkes, M. V.: *Time Sharing Computer Systems*, American Elsevier Publishing Co., New York, 1968.

本書は, タイムシェアリング・システムに興味を持つ方にすすめる.

Winograd, J., S. J. Morganstein, and R. Herman: "Simulation Studies

of a Virtual Memory, Time-Shared, Demand Paging Operating System," *Third ACM Symposium on Operating Systems Principles,* Stanford University, pp. 149-155, Oct. 1971.

Sim/61 と呼ばれる詳細なシステム・シミュレータについて述べている．これは，仮想記憶，タイムシェアリング，デマンド・ページング・システムの詳細なモデルを作る．このモデルは，タスク・スケジューリングに対する負荷と構成の研究およびアルゴリズムの評価に使用される，各種のシミュレーションによる実験が簡単に概説されている．本報告は，計算機の解析用の道具であるSimscript によって実行できる事例の優れた見本である．

Wulf, W. A.: "Performance Monitors for Multiprogramming Systems," *Second ACM Symposium on Operating Systems Principles,* Princeton University pp. 175-181, 1969.

60年代の中期以降，自己モニタリングによって自己調節を行なうシステムは計算機システムの技術者の目標であり，今日でも大部分の問題は研究中である．本報告では，システムおよびプロセスの動作の測定に基づいて多重プログラミング・システムの仕事量を改善する方法を提案している．ミックス・モニタリングによって資源の利用率を改善することが中心的な話題である．このマクロ／マイクロの動的な解析法は簡単な型式でB5500に組込まれている．

訳者あとがき

本書はオペレーティング・システムの原理と構造について体系的に述べたもので，大学上級生から大学院生向きの教科書として適当であり，また，ソフトウェア技術者の参考書でもある．本書の内容は姉妹書である**“システム・プログラム”**の第9章を発展させたものである．本書では，オペレーテング・システムが計算機システムに含まれた各種の資源の管理者であるという観点から論じている．

近年オペレーテング・システムの重要性が広く認識され，これに関する書物も幾つか見受けられるようになったが，中には単にオペレーティング・システムの機能を述べたものや，初歩的な概説を行なっているものも多く，本書のように，原理から実際の組込みにわたって一貫して述べたものは得られなかったといってよい．本書は，始めに資源の管理者であるという枠組をこしらえた上に，第7章の例題オペレーティング・システムや第9章の事例研究によって具体的なアルゴリズムにまで触れている点が大きな特徴である．読者の興味を一層引き立てるものと信じている．

訳者は前にドノバン教授の**システム・プログラム**（日本コンピュータ協会，1974）をも翻訳したが，オペレーテング・システムに関してさらに進んだ内容の教科書が発行されることを期待していたところであったので，早速，本書を翻訳することにしたのである．

M.I.T. の教育は名実ともに世界のトップ・レベルにあるが，本書の内容を演習や実習つきでみっちりやれば，実力のつくのはむしろ当然であるとも考えられる．読者はプログラムのリストも細部まで目をとおし，問題を解きながら進むのがよい．もちろん，細部に目をとられてしまうことなく，各章ごとに“いかに（how）”よりも“なぜ（why）”そうしているのかを反省して要約しつつ，重要な概念と，設計における問題点やどうでもよい選択事項とを，はっ

きりと区別しておかなければならない.

本書がオペレーティング・システムの正しい理解と発展のためにいささかでも役立つならば訳者の喜びとするところある.

本書の翻訳に当っては，できるだけ原著に忠実に行なうことを心がけながらも，直訳調とならないよう配慮し，単に翻訳しても意味の通じないところは適当に文章を補っておいた．また明らかに誤りであると考えられるところは訳者の責任で訂正しておいた.

本書を翻訳するに当り次の方々に大変お世話になった：

電々公社の杉山守氏および京都大学生の堀洋一君は原稿を精読され，きわめて適切な指摘をされた．京都大学大学院生金森邦雄君が行なつてくれた査読に対して感謝したい．原稿の浄書と校正は愚妻和子が，原稿の整理は中谷いつ子嬢が手伝ってくれた.

ここに厚く感謝し，その労をねぎらいたい.

1975年12月

訳　者

索　　引

I

J

K

L

M

N

T

U

V

W

Y

Z

<訳者略歴>

池田克夫（いけだかつお）

1937年　滋賀県に生まれる
1960年　京都大学工学部電子工学科卒業
1965年　京都大学大学院工学研究科博士課程終了
1965年　京都大学助手
1971年　京都大学工学部情報工学科助教授
　　　　文部省在外研究員にしてユタ大学および
　　　　マサチューセッツ工科大学に留学

現　在：筑波大学電子・情報工学系教授

オペレーティング・システム

1976年11月25日　初版発行
1978年3月20日　2版発行
1979年1月10日　3版発行
1980年11月30日　4版発行
1983年6月15日　5版発行

著　者　マドニック＆ドノバン
訳　者　池　田　克　夫
発行者　村　越　勝　弘
発行所　日本コンピュータ協会
東京都文京区本郷3丁目5—5
発売元　科学技術出版社
東京都文京区本郷3丁目5—5
〒113　電話（03）813—6920番